CSCD433
Advanced Networks
Fall 2012
Lecture 16

Raw vs. Cooked Sockets
Introduction

• Assigned task
  – Write a sniffer or port scanner program in Java
  – Use Raw Sockets
  – Have not done this before
  – So far, sockets in Java either
    • TCP or UDP based
  – In fact, Java does not have built-in support for Raw Sockets!!!
Motivation for Raw Sockets

- Usual Java Sockets do not fit all our needs
- Normal sockets lack some functionality

1. We cannot read/write ICMP or IGMP protocols with normal sockets
   - Ping tool cannot be written using normal sockets
2. Some Operating Systems do not process IPv4 protocols other than ICMP, IGMP, TCP or UDP

What if we have a proprietary protocol that we want to handle?

How do we send/receive data using that protocol?

Answer: Raw Sockets!!!
More Motivation for Raw Sockets

• Recall, CSCD330, can we send true ICMP packets in Java?
  – Not exactly.

  – There is this work-around
    InetAddress.getByName("192.168.1.1").isReachable(4000);
  – What does this do?
What does this do?

- InetAddress.getByName("192.168.1.1").isReachable(4000);
  - Does several things depending on OS and user permissions
    - **Linux/MacOS**
    - Linux/MacOS environment, no admin rights,
    - JVM tries to establish a TCP connection on port 7
      - Function returns true if TCP handshake is successful
    - With Superuser rights, correct ICMP request is sent and function returns true if an ICMP reply is received

- **Windows XP**
  - Windows XP environment, TCP handshake is used to test if machine is up, no matter if program has admin rights or not
Solution Using Raw Sockets

• There is a way in java, using jpcap library
  – Using jpcap, it is possible to assemble and send ICMP Packets
  – Code for a program to do this below

Question

- What's a Raw Socket in Java?
- How does it differ from a TCP/UDP Socket in Java?
Question

• What's a Raw Socket?

• How does it differ from a TCP/UDP Socket in Java?
  – Raw sockets offer you absolute control over data which is being sent or received through network
  – Raw socket allows an application direct access to lower-level communication protocols
  – Allows applications to bypass the TCP/IP Stack and have access to the entire packet + headers
A day in the life of Network Packet

Some slides courtesy of Vivek Ramachandran
The gory details .....
More Details on Raw sockets

- All Headers i.e. Ethernet, IP, TCP etc are stripped by network stack and only data is shipped to application layer
- We cannot modify packet headers of packets when they are sent out from our host
- **Is this a good thing?**
Injecting Packets

• If we could receive frames for all computers connected to our broadcast domain
  – Promiscuous Mode

• If we could get all the headers
  – Ethernet, TCP, IP etc from the network and analyze them

• Then, we could inject packets with custom headers and data into the network directly
Promiscuous Mode

• It is the “See All, Hear All” mode
  – Tells network driver to accept all packets irrespective of whom packets are addressed to
  • Used for Network Monitoring – both legal and illegal monitoring
  • We can do this by programmatically setting the IFF_PROMISC flag or
  • Using the ifconfig utility (ifconfig eth0 promisc)
Getting All headers - Sniffing

• Once we set interface to promiscuous mode we can get “full packets” with all the headers.
  – We can process these packets and extract data from it
  – Note we are receiving packets meant for all hosts
More on Promiscuous Mode

- Questions
- Under what circumstances can we see all packets on a LAN segment?
- Is promiscuous mode truly magic?
More on Promiscuous Mode

• Under what circumstances can we see all packets on a LAN segment?
• Is promiscuous mode truly magic?
• **Answer: NO**
  – Can see broadcast traffic
  – Can see all traffic if hosts are on a hub
  – Can see all traffic if one switch port is a mirror or spanning port
  – Can see all traffic, if card is able to go into promiscuous mode and LAN is wireless
Sending arbitrary packets – Packet Injection

- We “manufacture” our own packets and send it out on the network.
- Absolute power!!!
- Total network stack bypass
- Most active network monitoring tools and hacking tools use this.
- Dos attacks ? Syn Floods ? IP Spoofs ?
Raw Sockets – a closer look

Diagram:
- User land application
- Socket interface
- TCP/UDP processing
- IP processing
- Protocol family handling routines
- Network card driver
- LPF

Application
Raw Socket
What are raw sockets?

- Raw sockets provide a way to bypass whole network stack traversal of a packet and deliver it directly to an application.
- In Java, no way to get to RAW interfaces in OS kernels.
  - Two C libraries support standard way of interfacing to network cards:
    - Libpcap – Linux/MAC/Unix
    - Winpcap - Windows
Link Layer Packet Capture

• Stevens in his classic book, makes a distinction between Raw Sockets and capturing packets at the link layer
  • See, Unix Network Programming by R. Stevens, B. Fenner and A. Rudoff for details

• For our purposes, since Java doesn't have true RAW socket interface, only way we can capture raw traffic is through link layer packet capture
  • Jpcap running on top of libpcap library
Libpcap uses BPF and PF_Packet in Linux

BPF – Berkeley packet Filter

Gets copy after received
Gets copy before transmitted
Final Notes on Libpcap

- Winpcap same thing in windows
- Libpcap – Linux, unix and Mac
- TCPDump and Wireshark tools use Libpcap and Winpcap libraries
Jpcap

- Open source library for capturing and sending network packets from Java applications
- It provides facilities to:
  - Capture raw packets live from the wire.
  - Save captured packets to an offline file, and read captured packets from an offline file.
  - Automatically identify packet types and generate corresponding Java objects (for Ethernet, IPv4, IPv6, ARP/RARP, TCP, UDP, and ICMPv4 packets).
  - Filter packets according to user-specified rules before dispatching them to application and send raw packets to the network.
Java Jpcap

To get started, you must first create a class that implements the interface jpcap.JpcapHandler

```java
public class JpcapTip implements JpcapHandler {
    public void handlePacket(Packet packet) {
        System.out.println(packet);
    }
}
```
Java Jpcap

• You need to tell Jpcap which network device you want to listen to

• API provides jpcap.Jpcap.getDeviceList() method for this purpose

  ```java
  String[] devices = Jpcap.getDeviceList();
  ```

• Once you have a list of device names, you must choose one for listening:

  ```java
  String deviceName = devices[0];
  ```
After choosing a device, you open it for listening by using method `Jpcap.openDevice()`.

The `openDevice()` method requires four arguments:

1. Device name to be opened,
2. Maximum number of bytes to read from the device at one time,
3. Boolean value specifying whether to put the device into promiscuous mode,
4. Timeout value that will be used if you later call the `processPacket()` method.

```java
Jpcap jpcap = Jpcap.openDevice(deviceName, 1028, false, 10000);
```
Java Jpcap

Jpcap jpcap = Jpcap.openDevice(deviceName, 1028, false, 10000);

• The openDevice() method returns a reference to a Jpcap object that will be used for capturing
• Now have Jpcap instance, you can start listening by calling either processPacket() or loopPacket()
• Both of the methods take two arguments:
  – Maximum number of packets to capture can be -1 to indicate no limit
  – Instance of a class that implements JpcapHandler.
If you call processPacket(),
- Jpcap will capture packets until either the timeout specified in openDevice is exceeded
- or maximum number of packets specified has been reached

If you call loopPacket() 
- will capture packets until the maximum number of packets is reached or forever, if there is no maximum. The call looks like this:
```java
Jpcap.loopPacket(-1, new JpcapTip());
```
import jpcap.JpcapHandler;
import jpcap.Jpcap;
import jpcap.Packet;

public class JpcapTip implements JpcapHandler {
    public void handlePacket(Packet packet) {
        System.out.println(packet);
    }

    public static void main(String[] args) throws java.io.IOException {
        String[] devices = Jpcap.getDeviceList();
        for (int i = 0; i < devices.length; i++) {
            System.out.println(devices[i]);
        }
        String deviceName = devices[0];

        Jpcap jpcap = Jpcap.openDevice(deviceName, 1028, false, 1);
        jpcap.loopPacket(-1, new JpcapTip());
    }
}
JPCAP Example

The output of executing the test class looks like this:
It's shortened for space:

ARP REQUEST 00:06:5b:01:b2:4d(192.168.15.79)
  00:00:00:00:00:00(192.168.15.34)
ARP REQUEST 00:06:5b:01:b2:4d(192.168.15.79)
  00:00:00:00:00:00(192.168.15.34)
1052251329:525479 192.168.15.103->255.255.255.255 protocol(17)
  priority(0) hop(offset(0) ident(59244) UDP 1211  1211
Summary

• Raw sockets through jpcap allows capability not built into Java
  – Raw sockets are possible
  – Can write cool, new programs and access lower level protocols
  – Gives power to you, the programmer!!!
  – Allows for fun in manipulating packets!
Class over !!

New Assignment: Assignment 4, packet sniffer or port scanner