Lecture 23: Port Scanning, Vulnerability Scanning, Packet Sniffing, and Intrusion Detection

Lecture Notes on "Computer and Network Security"

by Avi Kak (kak@purdue.edu)

April 27, 2009

Goals:

- Port scanners
- The nmap port scanner
- Vulnerability scanners
- The Nessus vulnerability scanner
- Packet sniffers
- Intrusion detection

23.1: Port Scanning

- See Section 21.1 of Lecture 21 on the mapping between the ports and many of the standard and non-standard services. As mentioned there, each service provided by a computer monitors a specific port for incoming connection requests. There are 65,535 different possible ports on a machine.

- The main goal of port scanning is find out which ports are **open**, which are **closed**, and which are **filtered** (meaning that there is no reply at all from the remote host).

- If a port on a remote host is **open** for incoming connection requests and you send it a SYN packet, the remote host will respond back with a SYN+ACK packet (see Lecture 16 for a discussion of this).

- If a port on a remote host is **closed** and your computer sends it a SYN packet, the remote host will respond back with a RST packet (see Lecture 16 for a discussion of this).

- Let’s say a port on a remote host is **filtered** with something like an `iptables` based packet filter (see Lecture 18 slides) and your scanner sends it a SYN packet or an ICMP ping packet, you may not get back anything at all.

- A frequent goal of port scanning is to find out if a remote host is providing a service that is vulnerable to buffer overflow attack (see Lecture 21 slides for this attack).

- Port scanning may involve all of the 65,535 ports or only the ports that are well-known to provide services vulnerable to different security-related exploits.

23.2: Port Scanning with Calls to `connect()`

- The simplest type of a scan is made with a call to **`connect()`**. The manpage for this system call on Unix/Linux systems has the following prototype for this function:

  ```
  #include <sys/socket.h>

  int connect(int socketfd, const struct sockaddr *address, socklen_t address_len);
  ```

  where the parameter **`socketfd`** is the file descriptor associated with the internet socket constructed by the client (with a call to three-argument **`socket()`**), the pointer parameter **`address`** that points to a **`sockaddr`** structure that contains the IP address of the remote server, and the parameter **`address_len`** that specifies the length of the structure pointed to by the second argument.

- A call to **`connect()`** if successful completes a three-way handshake (that was described in Lecture 16) for a TCP connection with a server. The header file **`sys/socket.h`** include a number of definitions of structs needed for socket programming in C.

- When **`connect()`** is successful, it returns 0, otherwise it returns -1.

- In typical use of `connect()` for port scanning, if the connection succeeds, the port scanner immediately closes the connection (having ascertained that the port is open).

23.3: Port Scanning with TCP SYN Packets

- Scanning remote hosts with SYN packets is probably the most popular form of port scanning.

- As discussed at length in Lecture 16 when we talked about SYN flooding for DoS attacks, if your machine wants to open a TCP connection with another machine, your machine sends the remote machine a SYN packet. If the remote machine wants to respond positively to the connection request, it responds back with a SYN+ACK packet, that must then be acknowledged by your machine with an ACK packet.

- In a port scan based on SYN packets, the scanner machine sends out SYN packets to the different ports of a remote machine. When the scanner machine machine receives a SYN+ACK packet in return, the scanner can be sure that the port on the remote machine is open.

- In port scans based on SYN packets, the scanner never sends back the ACK packet to close any of the connections. So any connections that are created are always in half-open states, until of course they time out.

- Usually, instead of sending back the expected ACK packet, the scanner sends an RST packet to close the half-open connection.

23.4: The `nmap` Port Scanner

- `nmap` stands for "network map". This open-source scanner has been developed by Fyodor (see `http://insecure.org/`.).

- This is one of the most popular port scanners that runs on Unix/Linux machines.

- `nmap` is actually more than just a port scanner. In addition to listing the open ports on a network, it also tries to construct an inventory of all the services running in a network. It also tries to detect as to which operating system is running on each machine, etc.

- In addition to carrying out a TCP SYN scan, `nmap` can also carry out TCP `connect()` scans, UDP scans, ICMP scans, etc. **Regarding UDP scans, note that SYN is a TCP concept, so there is NO such thing as a UDP SYN scan. In a UDP scan, if a UDP packet is sent to a port that is NOT open, the remote machine will respond with an ICMP port unreachable message. So the absence of a returned message can be construed as a sign of an open UDP port. However, as you should know from Lecture 18, a packet filtering firewall at a remote machine may prevent**

the machine from responding with an ICMP error message even when a port is closed.

- As listed in the manpage, `nmap` comes with a large number of options for carrying out different kinds of security scans of a network. `nmap`. To give the reader a sense of the range of possibilities incorporated in these options, here is a partial description of the entries for the two options '-sP' and '-sV':

-sP : This option, also known as the "ping scanning" option, is **for ascertaining as to which machines are up in a network**. Under this option, Nmap sends out ICMP echo request packets to every IP address in a network. Hosts that respond are up. But this does not always work since many sites now block echo request packets. To get around this, nmap can also send a TCP ACK packet to (by default) port 80. If the remote machine responds with a RST back, then that machine is up. Another possibility is to send the remote machine a SYN packet and waiting for a RST or a SYN/ACK. **`For root users, nmap uses both the ICMP and ACK techniques in parallel.`** For non-root users, only the TCP `connect()` method is used.

-sV : This is also referred to as "Version Detection". After nmap figures out which TCP and/or UDP ports are open, it next tries to figure out what service is actually running at each of those ports. A file called `nmap-services-probes` is used to determine the best probes for detecting various services. In addition to determine the service protocol (http, ftp, ssh, telnet, etc.), nmap also tries to determine the application name (such as Apache httpd, ISC bind, Solaris telnetd, etc.), version number, etc.

- If nmap is compiled with OpenSSL support, it will connect to SSL servers to figure out the service listening behind the encryption.

- To carry out a port scan of your own machine, you could try (called as root)

  ```
  nmap -sS localhost
  ```

  The "-sS" option carries out a SYN scan. If you wanted to carry out an "aggressive" SYN scan of, say, `moonshine.ecn.purdue.edu`, you would call as root:

  ```
  nmap -sS -A moonshine.ecn.purdue.edu
  ```

  By the way, the "-sT" option carries out a TCP `connect()` scan.

- When I invoked nmap on localhost, I got the following result

```
Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2007-03-14 10:20 EDT
Interesting ports on localhost.localdomain (127.0.0.1):
(The 1648 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
22/tcp   open  ssh
25/tcp   open  smtp
53/tcp   open  domain
80/tcp   open  http
111/tcp  open  rpcbind
465/tcp  open  smtps
587/tcp  open  submission
631/tcp  open  ipp
814/tcp  open  unknown
953/tcp  open  rndc
1241/tcp open  nessus
3306/tcp open  mysql

Nmap run completed -- 1 IP address (1 host up) scanned in 0.381 seconds
```

- By default, **`nmap`** first pings a remote host in a network before scanning the host. The idea is that if the machine is down, why waste time by scanning all its ports. But since many sites now block/filter the ping echo request packets, this strategy may bypass machines that may otherwise be up in a network. To change this behavior, the following sort of a call to **`nmap`** may produce richer results (at the cost of slowing down a scan):

  ```
  nmap -sS -A -P0 moonshine.ecn.purdue.edu
  ```

  The '-P0' option (the second letter is 'zero') tells **`nmap`** to **not** use ping in order to decide whether a machine is up.

23.5: Vulnerability Scanners

- The terms *security scanner*, *vulnerability scanner*, and *security vulnerability scanner* all mean roughly the same thing. Any such "system" may also be called just a *scanner* in the context of network security. Vulnerability scanners frequently include port scanning.

- A vulnerability scanner scans a specified set of ports on a remote host and tries to test the service offered at each port for its known vulnerabilities.

- Be warned that an aggressive vulnerability scan may crash the machine you are testing. It is a scanner's job to connect to all possible services on all the open ports on a host. By the very nature of such a scan, a scanner will connect with the ports and test them out in quick succession. If the TCP engine on the machine is poorly written, the machine may get overwhelmed by the network demands created by the scanner and could simply crash. **That is why many sysadmins carry out security scans of their networks no more than once a month or even once a quarter.**

23.6: The Nessus Vulnerability Scanner

- According to the very useful web site "Top 100 Network Security Tools" (`http://sectools.org`), the source code for Nessus, which started out as an open-source project, was closed in 2005. Now you have to maintain a paid subscription to the company Tenable Computer Networks for the latest vulnerability signatures.

- Nessus is a *remote security scanner*, meaning that it is typically run on one machine to scan all the services offered by a remote machine in order to determine whether the latter is safeguarded against all known security exploits.

- According to the information posted at `http://www.nessus.org`: Nessus is the world's most popular vulnerability scanner that is used in over 75,000 organizations world-wide.

- The "Nessus" Project was started by Renaud Deraison in 1998. In 2002, Renaud co-founded Tenable Network Security with Ron Gula, creator of the Dragon Intrusion Detection System and Jack Huffard. Tenable Network Security is the owner, sole developer and licensor for the Nessus system.

- The Nessus vulnerability scanning system consists of a server and a client. They can reside in two separate machines.

- The server program is called **nessusd**. This is the program that "attacks" other machines in a network.

- The client program is called **nessus**. The client orchestrates the server, meaning that it tells the server as to what forms of attacks to launch and where to deposit the collected security information. The client packages different attack scenarios under different names so that you can use the same attack scenario on different machines or different attack scenarios on the same machine.

- While the server **nessusd** runs on a Unix/Linux machine, it is capable of carrying out a vulnerability scan of machines running other operating systems.

- The security tests for the Nessus system are written in a special scripting language called **Network Attack Scripting Lan-**

guage (NASL). Supposedly, NASL makes it easy to create new security tests.

- Each security test, written in NASL, consists of an **external plugin.** There are currently over 13,000 plugins available. New plugins are created as new security vulnerabilities are discovered. The command `nessus-update-plugins` can automatically update the database of plugins on your computer and do so on a regular basis.

- The client tells the server as to what category of plugins to use for the scan.

- Nessus can detect services even when they are running on ports other than the standard ports. That is, if the HTTP service is running at a port other than 80 or TELNET is running on a port other than port 23, Nessus can detect that fact and apply the applicable tests at those ports.

- Nessus has the ability to test SSLized services such as https, smtps, imaps, etc.

23.7: Configuring the `nessusd` Server

- You must first create one or multiple user accounts that will be used when a nessus client GUI connects with the nessus server to initiate a scan. This is done with the following command as root

  ```
  nessus-adduser
  ```

  This will ask for what authentication to use — **password authentication is the easiest**. It will also ask you for rules to be applied to the user. There is no great reason to use any user-specific rules at all for personal installations of Nessus. So leave this empty by entering `<ctrl-D>`. This will enable all possible permissions for the new user. For further information on this command, do

  ```
  man nessus-adduser
  ```

  There you can find out that to remove a user you should say as root

  ```
  nessus-rmuser
  ```

- This is the last step for server configuration. This step updates the plugins. Note that each plugin is based on a vulnerability signature:

```
nessus-update-plugins
```

By the way, this updating step only works if your server is registered with `http://www.nessus.org/register/`.

23.8: Installing the Nessus Client Software

- You need to install a nessus client before you can do any scanning with the server. As mentioned earlier, it is the client that controls the server (although it is the server that actually does the attacking and scanning).

- You have three choices for clients:
  - A command line client, `nessus`
  - A GUI based client in Linux. On a Ubuntu platform, when you execute just **nessus** in a command line, that brings up the GUI version of the nessus client.
  - A GUI based client in Windows. You run the server (meaning the scanner) on a Linux machine while the client is run remotely on a Windows machine. For this you must install on a Windows machine the dll's in

    ```
    nessuswx-1.4.5d-source.zip
    ```

23.9: Starting up and Configuring a Nessus Client

- Configuring a client means telling the client to create a particular attack scenario. But since it is the server that actually attacks a remote host, the server must be on before you can start putting together attack scenarios on the client.

- Therefore, you first start the **nessusd** server daemon by (as root)

  ```
  nessusd -D &
  ```

  where the ampersand is needed only if you want to server in the background. The '-D' option is for running the server as a daemon in the background. Do 'man nessusd' to all options for the server.

- The default port for the server daemon, `nessusd`, is 1241. This is the port the daemon will monitor for incoming connection requests from Nessus clients. To make sure that the server is monitoring this port, you can invoke:

  ```
  netstat -an | grep 1241
  ```

  It should return

  ```
  tcp  0    0    0.0.0.0:1241    0.0.0.0:*    LISTEN
  ```

or better yet by

```
netstat -tap | grep -i listen
```

- Now you are ready to start the client. It is possible to use a command-line invocation of the client by using syntax an example of which is shown below:

  ```
  nessus  -q  localhost  1241  <user>  <pwd>  <targets>  <results>
  ```

  where '-q' option is for running Nessus in the batch mode, and where `<user>` and `<pwd>` are the username and the password for the authorized user created with the `nessus-adduser` command when the server was first installed. The argument `<targets>` is the full pathname to the file that lists the symbolic hostnames of the target machines to be attacked and `<results>` is the name of the file in which the results of the scan can be dumped. Here is an example of this command line call

  ```
  nessus  -q  localhost  1241  ack2  ack2  nessus_targets  nessus.txt
  ```

  where I have assumed that `ack2` was a username created with the **nessus-adduser** command and that the password associated with this username is the same. In the above invocation, the name of the target machine to be scanned is in the file `nessus_targets` file and we want the scan results to be dumped in the file `nessus.txt`. *Note that by default the scanner will only check the first 15,000 ports.*

- Notwithstanding the above command-line syntax for firing up a client, you are more likely to use a GUI based client that we will now explain. On a Ubuntu machine, a GUI based client on a Linux machine can be brought up by:

  ```
  nessus &
  ```

  while you are logged in as root.

- The GUI will show the following tabs at the top:

  ```
  Nessusd host

  Plugins

  Credentials

  Scan Options

  Target

  User

  Prefs.

  KB
  ```

  The first of these, **Nessusd host** is the name of the machine on which the **nessusd** daemon server is running. If you running the client and the server on the same machine, **Nessusd host** can be set to `localhost`. The other information needed under the **Nessusd host** is the port number that the daemon server will be monitoring for connection requests from Nessus clients, the user login, and the user password. The user login name and the password must be what you created earlier with the **nessus-adduser** command. *You must enter the information required under the first tab and connect with the server in order to activate the other tabs.*

- Now let's talk about the second tab shown on the Nessus client GUI. This tab is for **Plugins**. Each plugin defines a separate test for a security vulnerability and currently there are over $13,000$ plugins. **The plugins are arranged into families of related tests.** The upper window under the **Plugin** tab will list the family names of the plugins. If you click on a family name, the lower window will show all the plugins in that family. If you click on one of the plugins in the lower window, a window will pop up explaining the nature of the plugin and what security threat the plugin represents. It is interesting to read the docs on the plugin for CGI related security holes. [For applying the Nessus scanner to a small home-based network, there would probably never be a reason to have all 13277 plugins turned on. Suppose all the machines in your network are either RedHat Linux and Windows, you'd want to disable the plugins that are meant for other operating system, such as for Mandrake Linux, MacOS, HP-UX, etc. To disable plugins either on an entire family basis or individually, you have to click on the respective plugin entry to highlight it, and to then scroll the display all the way to the right to see the checkbox for that entry. As you disable the not-needed plugins, note the count at the bottom of the Plugin Selection Display. After all the disabling I did, I was left with 5825 out of 13277 plugins.]

- I ignored the "Credentials" options. Apparently, you need it for the "Local Security Checks" feature of Nessus. When this feature is enabled, Nessus carries out an ssh login (using certificates) into each host on the network that has a 'nessus' account installed on it. Nessus then carries out a local security check on each host looking at the host from the inside. This security check consists of verifying that all the security related patches are installed and up-to-date.

- I'll next go through all the options for setting up a scan. These are under the **Scan Options** tab on the Nessus client GUI page. The scan options are

  1. **port range**: The default is 1 to 15000.

  2. I checked "Consider unscanned ports as closed". This makes scanning faster as it keeps Nessus from sending packets to ports that were not specified above.

  3. I went with the default of 20 for the number of hosts to test at one time. The Nessus server spawns that many scanner processes.

  4. I went with the default of 4 for the maximum number of security checks to be launched simultaneously. Each of the scanner processes mentioned above will launch 4 security check pro-

cesses (one for each plugin). [What that means is that with these settings, the Nessus server will launch a total of 80 processes.]

5. I ignored the "Path to CGI's"

6. I ignored the "Do a reverse lookup of the IP before testing it".

7. I ignored the "Optimize the tests". (See the Nessus client manual for why you may wish to disable it.)

8. I unchecked "Safe checks" and thus disabled it. But note that by disabling "Safe checks" you run the risk that some security checks may harm the host being attacked.

9. I ignored the "Designate hosts by their MAC address". The manual says that designating hosts by their MAC addresses can be useful in DHCP networks. I am not going for this option since I am hoping to specify the network hosts in my home network by their IP addresses. [The acronym MAC here stands for Media Access Control. Recall that in Lecture 15, we used the same acronym for Message Authentication Code.]

10. That brings us to the **Port Scanners** option under the **Scan Options** tab. Nessus lists the following port scanners to choose from:

- Netstat scanner (As described in Lecture 16, `netstat` is a utility for printing out information regarding network connections, routing tables, interface statistics, masquerade connections, and multicast memberships. Calling this a scanner makes sense for old platforms. For newer platforms, note that `netstat` cannot be invoked on a remote host; it can only be used to scan the local ports to see what relationship it is vis-a-vis the remote hosts.

- Ping the remote host.

- Nessus SNMP scanner (SNMP stands for Simple Network Management Protocol. It is the internet standard protocol for exchanging management information between management console applications and managed entities (hosts, routers, bridges, hubs). An SNMP scanner allows you to scan a list of hosts by carrying out ping, DNS, and SNMP queries. For each host queried, an SNMP query typically fetches the following information: whether or not the host is a router, the system description, current number of established TCP connections, the max number of TCP connections the host can support, the number of network interfaces on the host, etc.)

- **SYN scan** (Performs a fast SYN port scan. It achieves its speed by first computing the RTT (Round Trip Time) with ping and then using that info to quickly send SYN packets to the remote host. Needs the ping port scanner to be turned on.)

– Scan for **LaBrea Tarpitted Hosts**. Your `nessusd` server sends a bogus ACK and ACK+windowprobe packet to a host. Also sends a SYN packet to test for non-persisting LaBrea machines. **LaBrea is a program that creates a tarpit or, as some have called it, a "sticky honeypot". LaBrea takes over unused IP addresses on a network and creates "virtual machines" that answer to connection attempts. LaBrea answers those connection attempts in a way that causes the machine at the other end to get "stuck", sometimes for a very long time. The system uses IP aliasing to redirect an packet directed to an unused IP address in a network so that it can be processed by a machine with a legitimate IP address. So an incoming connection request, in the form of a SYN packet, to an unused IP address can be responded to with a SYN+ACK packet, which the remote intruder responds to with an ACK packet, thus completing the 3-way handshake. This is referred to as tarpitting unused IP addresses in a network because the remote intruder gets stuck, until the connections timeout, dealing with what look like open connections to the intruder.**

- That takes us to the **Target** tab on the Nessus client GUI. For the Target option, I entered

```
moonshine.ecn.purdue.edu
```

I did NOT enable "Perform a DNS Zone Transfer". My understanding is that, when enabled, it allows the Nessus client to figure out all of the hosts in a local network by downloading the zone information from a specified nameserver. Obviously, if the target nameserver is some external nameserver, the hosts returned will be what that nameserver is an authoritative nameserver for.

- I ignored the **User** tab on the client GUI.

- The next tab on the Nessus client GUI is **Prefs.** for Preferences. **The very long page under this tab gives you all kinds of options for controlling your scans.** Scroll down this page to see all the choices. You can also use this option to change the default values used for ports by some of the plugins. For example, the default port for SNMP attacks is 161. It can be changed by clicking on "SNMP Settings" and entering a new value in the panel underneath. Similarly, if HTTP access to a remote host requires a username and password, that can be supplied by clicking on "HTTP Login Page". **I did not change anything through this option.**

- The next tab on the Nessus client GUI is **KB**. The **KB** option stands for "Knowledge Base". Using this feature allows you to not disturb the users of your network by doing a daily portscan of a /24 network and to not waste the results of prior tests. The Knowledge Base is the list of information gathered about a trusted host. It contains the list of open ports, the OS on the host, and much more information.

- You are now ready to start scanning. Click on 'Start the Scan' at the bottom of the client GUI.

- After a scan is complete, a new window will pop up to display the report produced by the scan. You will have to click on the various items in this window to see the security holes and other vulnerabilities found by the scan. You can save the report for a permanent record by clicking on a button in the window.

23.10: Packet Sniffing

- A packet sniffer is a passive device (as opposed to a port or vulnerability scanners that by their nature are "active" systems).

- Packet sniffers are more formally known as **network analyzers** and **protocol analyzers**.

- The name **network analyzer** is justified by the fact that you can use a packet sniffer to *localize* a problem in a network. As an example, suppose that a packet sniffer says that the packets are indeed being put on the wire by the different hosts. Now if the network interface on a particular host is not seeing the packets, you can be a bit more sure that the problem may be with the network interface in question.

- The name **protocol analyzer** is justified by the fact that a packet sniffer could look inside the packets for a given service (especially the packets exchanged during handshaking and other such negotiations) and make sure that the packet composition is as specified in the RFC document for that service protocol.

- What makes packet sniffing such a potent tool is that a majority of LANs are based on the **shared Ethernet** notion. In a shared Ethernet, you can think of all of the computers in a LAN as being plugged into the same wire (notwithstanding appearances to the contrary). **So all the Ethernet interfaces on all the machines that are plugged into the same router will see all the packets.** On wireless LANs, all the interfaces on the same **channel** see all the packets meant for all of the hosts who have signed up for that channel.

- If you will recall from Lecture 16, it is the lowest layer of the TCP/IP protocol stack, the Link Layer, that actually puts the information on the wire. What is placed on the wire consists of data packets called **frames**. Each Ethernet interface gets a 48-bit address called the MAC address that is used to specify both the source and the destination of each frame. Even though each network interface in a LAN sees all the frames, any given interface normally would not accept a frame unless the destination MAC address corresponds to the interface. [Like its earlier usage in this lecture, the acronym MAC here stands for Media Access Control. Recall that in Lecture 15, we used the same acronym for Message Authentication Code.]

- Here is the structure of an Ethernet frame:

```
Preamble      D-addr     S-addr    Frame-Type      Data         CRC
               MAC        MAC

8 bytes       6 bytes    6 bytes    2 bytes                      4 bytes

<------------ MAC Header  ----------------->
<-------------------  maximum of 1518 bytes  ------------------->
```

where "D-addr" stands for destination address and "S-addr" for source address. The 8-byte "Preamble" field consists of alternating 1's and 0's for the first seven bytes and '10101011' for the last byte; its purpose is to announce the arrival of a new frame and to enable all receivers in a network to synchronize themselves to the incoming frame. The 2-byte "Type" field identifies the higher-level protocol (e.g., IP or ARP) contained in the data field. The "Type" field therefore tells us how to interpret the data field. The last field, the 4-byte CRC (Cyclic Redundancy Check) provides a mechanism for the detection of errors that might have occurred during transmission. *If an error is detected, the frame is simply dropped.*

- The minimum size of an Ethernet frame is 64 bytes (D-addr: 6 bytes, S-addr: 6 bytes, Frame Type: 2 bytes, Data: 46 bytes, CRC checksum: 4 bytes). Padding bytes must be added if the data itself consists of fewer than 46 bytes. The maximum size is limited to 1518 bytes. That is, the number of bytes in the data field must not exceed 1500 bytes.

- It is the Network Layer's job to map the destination IP address in an outgoing packet to the destination MAC address that is eventually inserted by the Link Layer (which is where the frames are created) in an outgoing frame that contains the packet.

- The system uses a protocol called the Address Resolution Protocol (ARP) to figure out the destination MAC address corresponding to the destination IP address. As a first step in this protocol, the system looks into the locally available ARP cache. If no MAC entry is found in this cache, the system broadcasts an ARP request for the needed MAC address. As this request propagates outbound toward the destination machine, either en-route gateway machine supplies the answer from its own ARP cache, or, eventually, the destination machine supplies the answer. The answer received is cached for a maximum of 2 minutes.

- A packet sniffer will accept all of the frames in the local Ethernet regardless of the destination MAC addresses in the individual frames.

- When a network interface does not discriminate between the incoming frames on the basis of the destination MAC address, we say the interface is operating in the **promiscuous mode**. **(You can easily get an interface to work in the promiscuous mode simply by in-**

voking '`ifconfg ethX promisc`' as root where ethX stands for the name of the interface (it would be something like eth0, eth1, etc.).)

- About the power of packet sniffers to "spy" on the users in a LAN, the **`dsniff`** packet sniffer contains the following utilities that can collect a lot of information on the users in a network

sshmitm : This can launch a man-in-the-middle attack on an SSH link. (See Lecture 9 for the Man-in-the-Middle attack). As mentioned earlier, basically the idea is to intercept the public keys being exchanged between two parties A and B wanting to establish an SSH connection. The attacker, X, that can eavesdrop on the communication between A and B with the help of a packet sniffer pretends to be B vis-a-vis A and A vis-a-vis B.

urlsnarf : From the sniffed packets, this utility extracts the URL's of all the web sites that the network users are visiting.

mailsnarf: This utility can track all the emails that the network users are receiving.

webspy : This utility can track a designated user's web surfing pattern in real-time.

and a few others

23.11: Packet Sniffing with `tcpdump`

- This is an open-source packet sniffer that comes bundled with all Linux distributions.

- You saw many examples in Lectures 16 and 17 where I used `tcpdump` to give demonstrations regarding the various aspects of TCP/IP and DNS. **The notes for those lectures include explanations for the more commonly used command-line options for `tcpdump`.**

- `tcpdump` uses the **pcap** API (in the form of the `libpcap` library) for packet capturing. (The Windows equivalent of `libpcap` is `WinCap`.)

- Check the **pcap** manpage in your Linux installation for more information about pcap. You will be surprised by how easy it is to create your own network analyzer with the **pcap** packet capture library.

- Here is an example of how I could use `tcpdump` on my Linux laptop:

  - First create a file for dumping all of the information that will be produced by `tcpdump`:

    ```
    touch tcpdumpfile
    chmod 600 tcpdumpfile
    ```

    where I have also made it inaccessible to all except myself as root.

  - Now invoke `tcpdump`:

    ```
    tcpdump -w tcpdumpfile
    ```

    This is where `tcpdump` begins to do its work. It will will print out a message saying as to which interface it is listening to.

  - After a while of data collection, now invoke

    ```
    strings tcpdumpfile | more
    ```

    This will print out all the strings, meaning sequences of characters delimited by nonprintable characters, in the tcpdumpfile. The function 'strings ' is in the binutils package.

  - For example, if you wanted to see your password in the dump file, you could invoke:

    ```
    strings tcpdumpfile | grep -i password
    ```

- Hit `<ctrl-c>` in the terminal window in which you started `tcpdump` to stop packet sniffing.

23.12: Snort for Intrusion Detection

- First a word about **intrusion detection** with a packet sniffer. Since it can examine every packet in a LAN, one would think that it should be possible to equip a packet sniffer with additional logic that would permit it to detect intrusions into the local network. Whereas it might be difficult to decide whether an intruder has broken into your network on the basis of just the packets entering any one machine **(this would be particulary the case for detecting botnet intrusions)**, if you can look at all the packets entering (and leaving) a network, you should be able to create a more powerful sentry for the whole network. Think of the analogy between posting a separate guard at each house in a city and allowing cops to roam free in the city so they can observe the goings-on both globally and locally.

- Snort does exactly that. By examining all the packets in a network and applying appropriate rulesets to them, it make do a powerful job of detecting intrusions.

- I think of `snort` as `tcpdump` on steroids. `snort` does everything that `tcpdump` does plus more. Like `tcpdump`, `snort` is an open-source command-line tool.

- What makes `snort` a popular choice is its easy-to-learn and easy-to-use rule language for intrusion detection. Just to get a sense of the range of attacks people have written rules for, here are the names of the rule files in `/etc/snort/rules` directory on my Ubuntu machine:

```
backdoor.rules                 community-web-iis.rules   pop2.rules
bad-traffic.rules              community-web-misc.rules  pop3.rules
chat.rules                     community-web-php.rules   porn.rules
community-bot.rules            ddos.rules                rpc.rules
community-deleted.rules        deleted.rules             rservices.rules
community-dos.rules            dns.rules                 scan.rules
community-exploit.rules        dos.rules                 shellcode.rules
community-ftp.rules            experimental.rules        smtp.rules
community-game.rules           exploit.rules             snmp.rules
community-icmp.rules           finger.rules              sql.rules
community-imap.rules           ftp.rules                 telnet.rules
community-inappropriate.rules  icmp-info.rules           tftp.rules
community-mail-client.rules    icmp.rules                virus.rules
community-misc.rules           imap.rules                web-attacks.rules
community-nntp.rules           info.rules                web-cgi.rules
community-oracle.rules         local.rules               web-client.rules
community-policy.rules         misc.rules                web-coldfusion.rules
community-sip.rules            multimedia.rules          web-frontpage.rules
community-smtp.rules           mysql.rules               web-iis.rules
community-sql-injection.rules  netbios.rules             web-misc.rules
community-virus.rules          nntp.rules                web-php.rules
community-web-attacks.rules    oracle.rules              x11.rules
community-web-cgi.rules        other-ids.rules
community-web-client.rules     p2p.rules
```

- **Let's now peek into some of the rule files that are used for intrusion detection**. Shown below are some beginning rules in the file `community-bot.rules`. **These rules look for botnets using popular bot software.** [A **botnet** is a typically a collection of compromised computers — usually called **zombies** or **bots** — working together under the control of their human handlers — frequently called **bot herders** — who may use the botnet to spew out malware such as spam, spyware, etc. It makes it more difficult to track down malware if it seems to emanate randomly from a large network of zombies.] A bot herder typically sets up an IRC (Internet Relay Chat) channel for instant communications with the bots under his/her control. Therefore, the beginning of the ruleset shown below focuses on the IRC traffic in a network. [Although it is relatively trivial to set up a chat server (for example, see Chapter 19 of my PwO book for C++ and Java examples and Chapter 15 of my SwO book for Perl and Python examples), what makes IRC different is that one IRC server can connect with other IRC servers to expand the IRC network. Ideally, when inter-server hookups are allowed, the servers operate in a tree topology in which the messages are routed only through the branches that are necessary to serve all the clients but with every server aware of the state of the network. IRC also allows for private client-to-client messaging and for private individual-to-group link-ups. **That should explain why bot herders like IRC.** Joining an IRC chat does not require a log-in, but it does require a nickname (frequently abbreviated as just `nick` in IRC jargon).]

```
# The following rule merely looks for IRC traffic on any TCP port (by detecting NICK change
# events, which occur at the beginning of the session) and sets the is_proto_irc flowbit.
# It does not actually generate any alerts itself:
alert tcp any any -> any any (msg:"COMMUNITY BOT IRC Traffic Detected By Nick Change"; \
flow: to_server,established; content:"NICK "; nocase; offset: 0; depth: 5; flowbits:set,\
community_is_proto_irc; flowbits: noalert; classtype:misc-activity; sid:100000240; rev:3;)

# Using the aforementioned is_proto_irc flowbits, do some IRC checks.  This one looks for
# IRC servers running on the $HOME_NET
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"COMMUNITY BOT Internal IRC server detected"; \
flow: to_server,established; flowbits:isset,community_is_proto_irc; classtype: policy-violation; \
sid:100000241; rev:2;)


# These rules look for specific Agobot/PhatBot commands on an IRC session
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"COMMUNITY BOT Agobot/PhatBot bot.about \
command"; flow: established; flowbits:isset,community_is_proto_irc; content:"bot.about"; \
classtype: trojan-activity; sid:100000242; rev:2;)


alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"COMMUNITY BOT Agobot/PhatBot bot.die command";
flow: established; flowbits:isset,community_is_proto_irc; content:"bot.die"; classtype:
```

```
trojan-activity; sid:100000243; rev:2;)
....
....
....
```

- Next let us peek into the file `community-virus.rules`. Here are the first three rules, meant for detecting the viruses Dabber (at two different ports) and BlackWorm.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 5554 (msg:"COMMUNITY VIRUS Dabber PORT overflow \
attempt port 5554"; flow:to_server,established,no_stream; content:"PORT"; nocase; isdataat:100,\
relative; pcre:"/^PORT\s[^\n]{100}/smi"; reference:MCAFEE,125300; classtype:attempted-admin; \
sid:100000110; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 1023 (msg:"COMMUNITY VIRUS Dabber PORT overflow \
attempt port 1023"; flow:to_server,established,no_stream; content:"PORT"; nocase; isdataat:100,\
relative; pcre:"/^PORT\s[^\n]{100}/smi"; reference:MCAFEE,125300; classtype:attempted-admin; \
sid:100000111; rev:1;)

alert tcp $HOME_NET any -> 207.172.16.155 80 (msg:"COMMUNITY VIRUS Possible BlackWorm or \
Nymex infected host"; flow:to_server,established; uricontent:"/cgi-bin/Count.cgi?df=765247"; referenc
Win32%2fMywife.E%40mm; reference:url,cme.mitre.org/data/list.html#24; reference:url,isc.\
sans.org/blackworm; classtype:trojan-activity; sid:100000226; rev:2;)

....
....
```

- It is easy to install **`snort`** through your Synaptic Packet Manager, but be warned that the installation does not run to completion without additional intervention by you. Before telling you what that intervention is, the installation will place the executable in `/usr/sbin/snort`, the start/stop/restart script in `/etc/init.d/snot`, and the config files in the `/etc/snort/`

directory. As you'd expect the documentation is placed in the `/usr/share/doc/snort/` directory. Please read the various `README` files in this directory before completing the installation. Some of these `README` files are compressed; so you will have to use a command like

```
    zcat README.Debian.gz | more
```

to see what the instructions are. As you will find out from these `README` files, a full installation of `snort` requires that you also install a database server like `MySQL` or `PostgreSQL`. **But if you want to just have fun with `snort` as you are becoming familiar with the tool, it is not necessary to do so.** You just need to make sure that you delete the zero-content file named `db-pending-config` from the `/etc/snort/` directory.

- The syntax for writing the intrusion detection rules is explained in the file `/usr/share/doc/snort/snort_rules.html`.

- Your main config file is `/etc/snort/snort.conf`, but it should be good enough as it is for an initial introduction to the system.

- Once you get `snort` going, try the following command lines as root:

```
    snort -v -i eth0                // will see ALL packets visible
```

```
                               // to the eth0 interface

                               // the -v option is for verbose
                               // it slows down snort and it can lose
                               // packets with -v

snort -d -e -a -i eth0         // will also show you data in packets
                               // -d option is for data, -e is for
                               // link-layer packets, -a for ARP

snort -dea -i eth0             // a compressed form of the above


snort -d -i eth0 -l logile -h 192.168.1.0/24     // will scan your home
                                                 // LAN and dump info
                                                 // into the log file

snort -d -i eth0 -l logfile -c rule-file      // will dump all of the
                                              // info into the logfile
                                              // but only for packets
                                              // that trigger the rules
```

Do ‘`man snort`’ to see all the options.

- If instead of the above command lines, you start up snort with (as root, of course):

  ```
  /etc/init.d/snort start
  ```

  and then if you do `ps ax | grep snort`, you will discover that this automatic start is equivalent to the following command line invocation:

  ```
  snort -m 027 -D -d -l /var/log/snort -u snort -g snort -c /etc/snort/snort.conf\
  -S HOME_NET=[192.168.0.0/16] -i eth0
  ```

  assuming you are connected to a home LAN (192.168.1.0/24). Note the `-c` option here. In this case, this option points to the config file itself, meaning in general all the rule files pointed to by the config file.

- You can customize how `snort` works for each separate interface by writing a config file specific to that interface. The naming convention for such files is `/etc/snort/snort.$INTERFACE.conf`

- Some of the source code in `snort` is based directly on `tcpdump`.

- Martin Roesch is the force behind the development of Snort. It is now maintained by his company Sourcefire. The main website for Snort is `http://www.snort.org`. The main manual for the system is `snort_manual.pdf` (it did not land in my computer with the installation).

23.13: Packet Sniffing with `wireshark` (formerly `ethereal`)

- **`wireshark`** is a packet sniffer that, as far as the packet sniffing is concerned, work pretty much the same way as **`tcpdump`**. (It also uses the **pcap** library.) What makes **`wireshark`** special is its GUI front end that makes it extremely easy to analyze the packets.

- As you play with wireshark, you will soon realize the importance of a GUI based interface for understanding the packets and analyzing their content in your network. As but one example of the ease made possible by the GUI frontend, suppose you have located a suspicious packet and now you want to look at the rest of the packets in just that TCP stream. With wireshark, all you have to do is to click on that packet and turn on "follow TCP stream feature". Subsequently, you will only see the packets in that stream. The packets you will see will include resend packets and ICMP error message packets relevant to that stream.

- With a standard install of the packages, you can bring up the wireshark GUI by

```
wireshark
```

Yes, you can call **`wireshark`** with a large number of options to customize its behavior, but it is better to use the GUI itself for that purpose. So call **`wireshark`** without any options as shown above.

- The **`wireshark`** user's manual (HTML) is readily accessible through the "Help" menu button at the top of the GUI.

- To get started with sniffing, you could start by clicking on "capture". This will bring up a dialog window that will show all of the network interfaces on your machine. Click on "Start" for the interface you want to sniff on. Actually, instead click on the "Options" for the interface and click on "Start" through the resulting dialog window where you can name the file in which the packets will be dumped.

- You can stop sniffing at any time by clicking on the second-row icon with a little red 'x' on it.

- Wireshark understand 837 different protocols. You can see the list under "Help" menu button. It is instructive to scroll down this list if only to get a sense of how varied and diverse the world internet communications has become.

- Wireshark gives you three views of each packet:
  - A one line summary that looks like

```
Packet  Time      Source        Destination    Protocol   Info
Number
------------------------------------------------------------------

 1   1.018394   128.46.144.10  192.168.1.100  TCP   SSH > 33824 [RST,ACK] ..
```

  - A display in the middle part of the GUI showing further details on the packet selected. Suppose I select the above packet by clicking on it, I could see something like the following in this "details" display:

```
Frame 1  (54 bytes on the wire, 54 bytes captured)
Ethernet II, Src: Cisco-Li_6f:a8:db  (00:18:39:6f:a8:db), Dst: ..........
Internet Protocol:  Src: 128.46.144.10 (128.46.144.10)   Dst: .......
Transmission Control Protocol:  Src Port: ssh (22), Dst Port: 33824 ....
```

  - The lowest part of the GUI shows the hexdump for the packet.

- Note that wireshark will set the local Ethernet interface to promiscuous mode so that it can see all the Ethernet frames.