INFORMATION SECURITY FALL 2018
PROJECT: PORT SCAN DETECTION

Objectives: For this project you will complete the following:

- Use LIBPCAP or Python equivalent to monitor Ethernet activity on your Linux host.
- Detect port scans of various types in provided PCAP files.

This project requires the use of a Linux computer. Linux running in VMWare will work.

LIBPCAP

LIBPCAP is a library used to capture network transactions from a network card. It is used by Wireshark. We will use it to capture network transactions and print information about those transactions. LIBPCAP can open a live connection to your NIC card or can run from files. We will use both. The difference is actually trivial.

Here are some useful functions:

Function	Description
pcap_lookupdev	Returns a pointer to a string giving the name of a network device suitable for use with pcap_open_live.
pcap_open_live	Creates live connection to the network device. Returns pointer to pcap_t which is used like a file handle.
pcap_loop	Calls a call back function (the pcap handler routine) specified by the user every time a Ethernet packet is captured.
pcap_open_offline	Opens a ".pcap" file. Returns same handle as pcap_open_live.
pcap_dispatch	Similar to pcap_loop. Used to process all Ethernet packets found in the open ".pcap" file. Calls the pcap handler routine each time a Ethernet packet is found in the open file.
ntohs	Converts an unsigned short integer from network byte order to host byte order. This handles endianess so you don't have to. If the network endianess is the same as your host's endianess this does nothing. If they are different this converts the number for you.
ether_ntoa	Converts a Ethernet MAC address from network format (hex) to a string which looks like your typical MAC address (eg. 01:23:45:67:89:ab). Use this to print the MAC address.
inet_ntoa	Converts a network address in a struct in_addr to a dots-and-numbers format string. (eg. 192.168.1.1). Use this to print the IP address.

Here are some useful structs:

Struct	Description

struct ether_header	Ethernet header –
	`struct ether_header`
	`{`
	`  u_int8_t  ether_dhost[ETH_ALEN];      /* destination eth addr */`
	`  u_int8_t  ether_shost[ETH_ALEN];      /* source ether addr    */`
	`  u_int16_t ether_type;                 /* packet type ID field */`
	`} __attribute__ ((__packed__));`
	
	`ether_type tells you what is in the packet:`
	
	#define ETHERTYPE_IP 0x0800 /* IP */
	#define ETHERTYPE_ARP 0x0806 /* Address resolution */
	#define ETHERTYPE_IPV6 0x86dd /* IP protocol version 6 */
	
	Available in net/ethernet.h which is included in netinet/if_ether.h.

struct ip	IP header –
	`struct ip {`
	`   unsigned int   ip_hl:4; /* both fields are 4 bits */`
	`   unsigned int   ip_v:4;`
	`   uint8_t        ip_tos;`
	`   uint16_t       ip_len;`
	`   uint16_t       ip_id;`
	`   uint16_t       ip_off;`
	`   uint8_t        ip_ttl;`
	`   uint8_t        ip_p;`
	`   uint16_t       ip_sum;`
	`   struct in_addr ip_src;`
	`   struct in_addr ip_dst;`
	`};`
	
	`ip_src - source IP address`
	`ip_dst - destination IP address`
	
	Available in netinet/ip.h.

Struct	Description
struct tcphdr	TCP header -

```
struct tcphdr {
        unsigned short source;
        unsigned short dest;
        unsigned long seq;
        unsigned long ack_seq;
        #  if __BYTE_ORDER == __LITTLE_ENDIAN
        unsigned short res1:4;
        unsigned short doff:4;
        unsigned short fin:1;
        unsigned short syn:1;
        unsigned short rst:1;
        unsigned short psh:1;
        unsigned short ack:1;
        unsigned short urg:1;
        unsigned short res2:2;
        #  elif __BYTE_ORDER == __BIG_ENDIAN
        unsigned short doff:4;
        unsigned short res1:4;
        unsigned short res2:2;
        unsigned short urg:1;
        unsigned short ack:1;
        unsigned short psh:1;
        unsigned short rst:1;
        unsigned short syn:1;
        unsigned short fin:1;
        #  endif
        unsigned short window;
        unsigned short check;
        unsigned short urg_ptr;
};
```

source – source port
dest – destination port
seq – sequence number

syn, ack, fin, rst, psh, urg – flags

Available in netinet/tcp.h.

Struct	Description

struct udphdr	UDP header –
	`struct udphdr`
	`{`
	`  u_int16_t source;`
	`  u_int16_t dest;`
	`  u_int16_t len;`
	`  u_int16_t check;`
	`};`
	
	source – source port
	dest – destination port
	len – length
	
	Available in netinet/udp.h.

struct icmphdr	ICMP header –
	struct icmphdr
	{
	  u_int8_t type; /* message type */
	  u_int8_t code; /* type sub-code */
	  u_int16_t checksum;
	  union
	  {
	    struct
	    {
	      u_int16_t id;
	      u_int16_t sequence;
	    } echo; /* echo datagram */
	    u_int32_t gateway; /* gateway address */
	    struct
	    {
	      u_int16_t __unused;
	      u_int16_t mtu;
	    } frag; /* path mtu discovery */
	  } un;
	};
	
	type – indicates the ICMP message type. ICMP echo request is type 8.
	
	Available in netinet/ip_icmp.h.

pcap_t	Think of this as a file handle pointing to either an open ".pcap" file or a live connection to the NIC.

The routines *pcap_loop* and *pcap_dispatch* call a function that you define when a Ethernet packet is captured. This function is called the handler. The handler must have the following port list:

void handler(u_char *args, const struct pcap_pkthdr *header, const u_char *packet);

You define the handler and provide a reference to it when you call *pcap_loop* and *pcap_dispatch.* The packet variable in the handler port list is a pointer to the Ethernet header and its data fields.

Remember Ethernet packets encapsulate IP packets. IP packets encapsulate ICMP, TCP or UDP packets.

In memory the packet pointer points to memory which looks like:

Address	What is there?
packet	Ethernet Header
packet + sizeof(Ethernet header)	Ethernet Data = IP Header
packet + sizeof(Ethernet header) + size of(IP header)	IP Data = TCP, UDP, or ICMP header
packet + sizeof(Ethernet header) + size of(IP header) + sizeof(TCP or UDP header)	Data

So you can use pointer arithmetic shown above to create pointers to the various headers. From there you can type cast that pointer and assign it to a pointer to a struct ether_header, struct ip, struct tcphdr, or struct udphdr. From there you can easily print out the required MAC and IP addresses and port numbers.

GETTING LIBPCAP

You need to install the following libraries. The easiest way to do this is with Fedora's yum command.

libpcap.i586	sudo yum install libpcap.i586
libpcap-devel.i586	sudo yum install libpcap-devel.i586

If you are using a different Linux distribution you will need to look up how to install these packages and they may have a different name.

If you are programming with Python please use the web to select an appropriate library and follow install instructions.

SUDO PRIVILEGES

You will need SUDO privileges on your Linux host to be able to monitor the Ethernet port.

1. At the command prompt type: *su* ↲
2. Enter your root password.
3. edit the file: /etc/sudoers
4. On the last line of the file:
    a. add: "username ALL=(ALL) ALL"
    b. Substitute your user name for "username".
    c. Replace the path with the path to your executable.
5. At the command prompt type: *exit* ↲

To run your executable:

1. type "sudo ./psd.exe ↲"
   sudo will ask for your password the first time you run it and periodically thereafter.

That /etc/sudoers line gives your user account unlimited root powers. Remember, Padawan, with great power comes great responsibility.

Conventions
Name your executable “psd.exe” or “psd.py”. Name your source code “psd.c” or “psd.py”.

psd.exe should be run with either an input file or live.

from file – % psd.exe filename↲

Use argc to know if you have a filename or not.

How to compile:

1. For C programs, include a file named “comp” which includes the following lines:

```
#!/bin/csh
gcc -I/usr/lib -lpcap -o psd.exe psd.c
```

I will run the following commands to check your work:

1. ./comp
2. ./psd.exe <filename>

or

1. python psd.py <filename>

REQUIREMENTS

Null Scan	You program should identify all NULL scan attempts in the file null_scan.pcap. List the IP source address of the scanner and all ports scanned. Your program should not find any NULL scans in the noscan.pcap file.
Xmas Scan	You program should identify all XMAS scan attempts in the file xmas_scan.pcap. List the IP source address of the scanner and all ports scanned. Your program should not find any XMAS scans in the noscan.pcap file.
UDP Scan	You program should identify all UDP scan attempts in the file udp_scan.pcap. List the IP source address of the scanner and all ports scanned. Your program should not find any UDP scans in the noscan.pcap file.
IMCP	You program should identify all ICMP Echo request packets and print the source/dest IP address related to that ping. Do not print info for ICMP Echo replies.
Half-Open Scan	You program should identify all half-open scan attempts in the file halfopen.pcap. List the IP source address of the scanner and all ports scanned. Your program should not find any half-open scans in the noscan.pcap file.

GRADING

Item	Points
Code compiles/runs	20
Detect NULL	20
Detect XMAS	10
Detect UDP	20
Detect ICMP Echo packets	20
Detect Half-Open Scans	10
Total	100