**Software Architecture :**

Our software is divided into 3 main parts: bind, listen and send.

Flow diagram:

Main(entry)

Bind(Initialize)

listen

send

Show result

1. Bind

We have a lots of network adapter in our computer. We should firstly choose one to use in this software and set the target address, as well as the port range to be scanned.

We use a windows API and a famous lib named winpcap to get the devices and bind one.

Function details:

GetLocalAdapter():

get the list of the devices and ask the user to choose one

GetMacByArp():

Using arp protocol and ip address to find the mac address

pcap\_open\_live():

bind to an adaptor and set some basic information of the adaptor.

1. Listen

We use a function--pcap\_loop to receive any packets in our network adapter in another thread, which makes it faster to our software. After we receive a packet, we call another function to handle this packet. In this function we should first check whether this packet is a reply of our packets. We use the port 4139 in our computer, so the reply must has the dst\_port of 4139. We use this to check whether this packet is a reply of our packets. If this packet is ours, we then check if it is a rst reply or a ack/syn reply. A rst reply means that the port is closed and the ack/syn reply means it’s open.

Function details:

CreateThread():

Create a new thread to listen and handle the packet

pcap\_loop():

listen to the port and copy every packet to a pointer

print\_ack\_syn ():

handle the packet and show the result

1. Send

In this part, we use the pcap\_sendpacket(…) function to send an [elaborately](app:ds:elaborately) TCP packet. We also set the IP header and the Ethernet header. So we should use the ARP protocol to find the MAC address of the dst.

**Other Technical:**

We choose the C language and the winpcap lib to support our software. The reason why we choose the C language is that it is easy to use a pointer to format a packet which is binary data in the memory. That makes us easy to check different field in the packet. What’s more, the C language is friendly to use the winpcap lib.

After the service pack 2 of Windows XP, we could not use the raw socket to send some packet in the windows system for some security reasons. A suggested solution is to use the winpcap. In the winpcap, it has its own driver as well as own API for the user mode. In this way, we can still use something like a raw socket to send the packets.

**What we already Have:**

We are able to do

1. syn scan.
2. port service discovery.

**SYN Scan:**

So there comes a better way to scan, that called SYN scan. Also known as half-open scan.

It doesn’t need to establish a full TCP connect() and can get information about the port that been scanned. It’s very similar to the TCP scan, but it will tear down the connection immediately when receive a SYN/ACK instead of sending ACK responses back.

So the Scan will look like the diagram below if the port is open:

Host A(status): Host B(status)

Send SYN(seq =x )

Receive SYN(seq = x)

Send SYN/ACK(seq=y,ack=x+1)

Receive SYN/ACK(seq=y,ack=x+1)

Receive RST, connection shut down.

Send RST

And if the port is closed, we will receive a RST/ACK:

Host A (status): Host B (status)

Send SYN(seq =x )

Receive SYN(seq = x)

Send RST/ACK

Receive RST/ACK, then shut down the connection

And if we do the scan in this way, we will always be unlogged by the connection with basic IDS. We can also get the fairly positive information about the port.

And we do save a lot of time by tearing down the connection immediately instead of sending ACK responses. And it has reliable capability to recognize the port that if it’s open or not.

Though the standard Intrusion detection systems, but it was used to avoid the IDS.

**Advantage:**

Due to this type of scan doesn’t need to do the full of TCP connect().

* It’s much more quicker than full TCP connect().
* It’s reliable for recognize a port.
* It can avoid the basic IDS log.
* It needn’t to do the three handshakes.

**Disadvantage:**

* It would cause problem for some network stacks, particular for the simple device, such as printer.
* Only if you have root privilege, you can create yourself SYN pocket.
* Nowadays, rule sets block so many SYN scan attempt.

**Port Service discovery:**

We have a enum type of structure to store the popular service port, such as 25 for smtp and 80 for http.

Once we have detected the number of the port,then we could use the port num to compare with that structure then we will know the service supplied by that port.

**Enhancement for our TCP Scanner:**

Nowadays, many firewall can log syn, so we need an advanced method to avoid logged.

(1)FIN Scan

(2)ACK Scan

These three scans can enhance our TCP Scanner.

**Nmap do have that kinds of Scan:**

* FIN scan(No use for windows): Exploit the theory that once a port is closed it will tend to return a RST. If the port is opened it will ignored the package that have something wrong(reference the RFC 793 page 43).

(reference <http://nmap.org/book/man-port-scanning-basics.html> )

If the port is CLOSE:

Host A (status): Host B (status)

Receive FIN

Send FIN

Send a RST

Receive RST

If the port is open:

Host A (status): Host B (status)

Receive FIN

Send FIN

No reaction.

The Host B tend to throws the Package away.

If we once get the ICMP package, then we could not judge the status about that port

Host A (status): Host B (status)

Receive FIN

Send FIN

Send a ICMP error message

Receive ICMP error message

* ACK scan: It’s an advanced scan type, it does not actually give the information about whether the port is open or closed. But it can use to find the port that unfiltered. It’s used to map firewall rulesets,classifies ports into this state.

Whether it is open or not it will return RST package. Then analyze the package received.

(reference <http://nmap.org/book/man-port-scanning-basics.html> )

Host A (status): Host B (status)

Receive ACK

Send ACK

Send RST

Receive RST