Digital Forensics
Network Forensics
Lecture 2

Port Scanning

Outline

- Introduction
- Port scanning techniques
  - SYN scan
  - FIN, X-mas and NULL scans
  - Spoofing Decoy
  - Idle scanning
  - Proactive Defense

Introduction I

What is port scanning ?

A way of figuring out which ports are listening and accepting connections.

Why do hackers scan for open ports ?

Since most services run on standard ports, information obtained from port scanning can be used to determine which services are running.

Introduction II

A simple form of port scanning

Trying to open TCP connections to every possible port on the target system.

Disadvantages

- Noisy and detectable
- When connections are established, services will normally log the IP address.
  - Clever techniques have been invented to avoid this.
  - An open-source tool that implements such clever techniques in nmap.

Stealth SYN (half-open) scan

- In TCP/IP handshake, When a full connection is made:
  1. A SYN packet is sent.
  2. SYN/ACK packet is sent back.
  3. ACK packet is returned to complete the handshake and open the connection.

- A SYN scan **does not** complete the handshake, so a full connection is never opened.
  - Only the initial SYN packet is sent and the response is examined.

SYN scan using nmap

- Using nmap, a SYN scan can be performed using the command-line option –sS.
- nmap must be run as root, since it is not using standards sockets and needs raw network access.

```
reader@hacking:~/booksrc $ sudo nmap –sS 192.268.42.72

Starting Nmap 4.20 ( http://insecure.org ) at 2007-05-29 09:19 PDT
Interesting ports on 192.168.42.72:
Not shown: 1696 closed ports
PORT    STATE   SERVICE
22/tcp  open    ssh

Nmap finished: 1 IP address ( 1 host up ) scanned in 0.094 seconds
```

FIN, X-mas and NULL scan I

- Another collection of techniques for stealth port scanning evolved: SIN, X-mas and NULL scans.

Mechanism

1. send a nonsensical packet to every port on the target system.
2. If a port is listening these packet just get ignored.
3. If port is closed and implementation follows protocol (RFC 793), an RST packet will be sent.

- The mechanism can be used to detect which ports are accepting connections, without actually opening any connection.

FIN, X-mas and NULL scans II

- **FIN scan** sends a FIN packet
- **X-mas scan** sends a packet with FINURG and PUSH turned on (called X-mas since the flags are lit up like Christmas tree)
- **NULL scan** sends a packet with no TCP flags set.

Disadvantage

- While this type of scans are stealthier, they can also be unreliable.
  - Example: Microsoft's implementation of TCP does not send RST packets like it should, making this form of scanning ineffective.

Using nmap for FIN, X-mas and NULL scans

- Using nmap FIN, X-mas and NULL scans can be performed using –sF , -sX and –sN options.
- Their outputs are basically the same as SYN scan.

Decoy Node

Decoy Node

A kind of node in network, whose only purpose is to mislead the adversary.

- If the real and decoy nodes have valid IP addresses that are visible to an external adversary, then the adversary may mount attacks on decoy nodes instead of the real node, wasting the resources of the adversary and providing information to the system regarding the goals and capabilities of the adversary.

Clark, Andrew, et al. "A game-theoretic approach to IP address randomization in decoy-based cyber defense." *International Conference on Decision and Game Theory for Security*. Springer, Cham, 2015.

Spoofing Decoy

- Spoofing decoy technique spoofs connections from various decoy node IP addresses in between each real port-scanning connection.

- The spoofed decoy addresses must use real IP addresses of live hosts; otherwise the target may be accidentally SYN flooded.
- The following sample nmap command scans the IP 192.168.42.72, using 192.168. 42.10 and 192.168.42.11 as decoys.

```
reader@hacking:~/booksrc $ sudo nmap -D 192.168. 42.10, 192.168.42.11 192.168.42.72
```

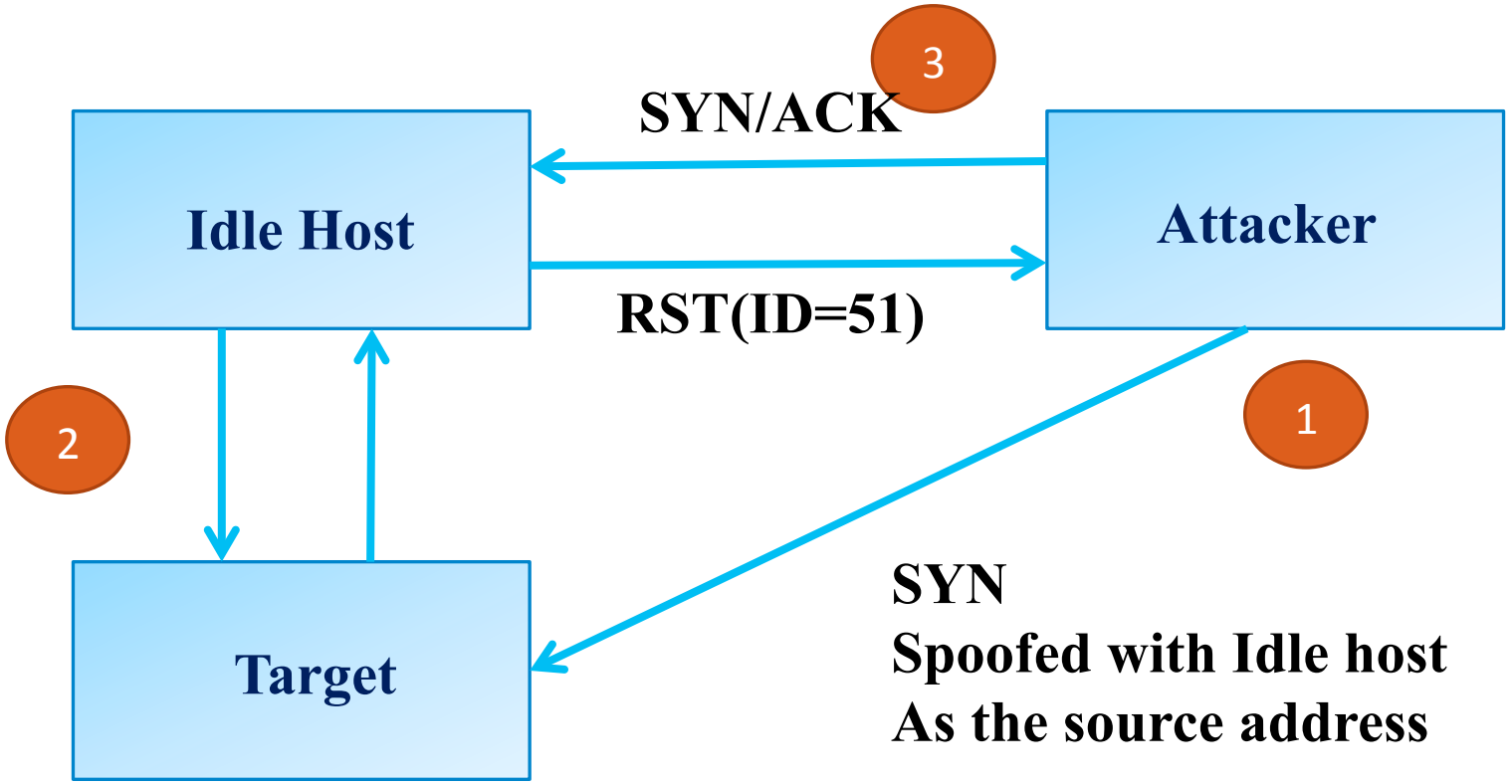
Idle Scanning

- A way to scan a target using spoofed packets from an idle host, by observing changes in the idle host.

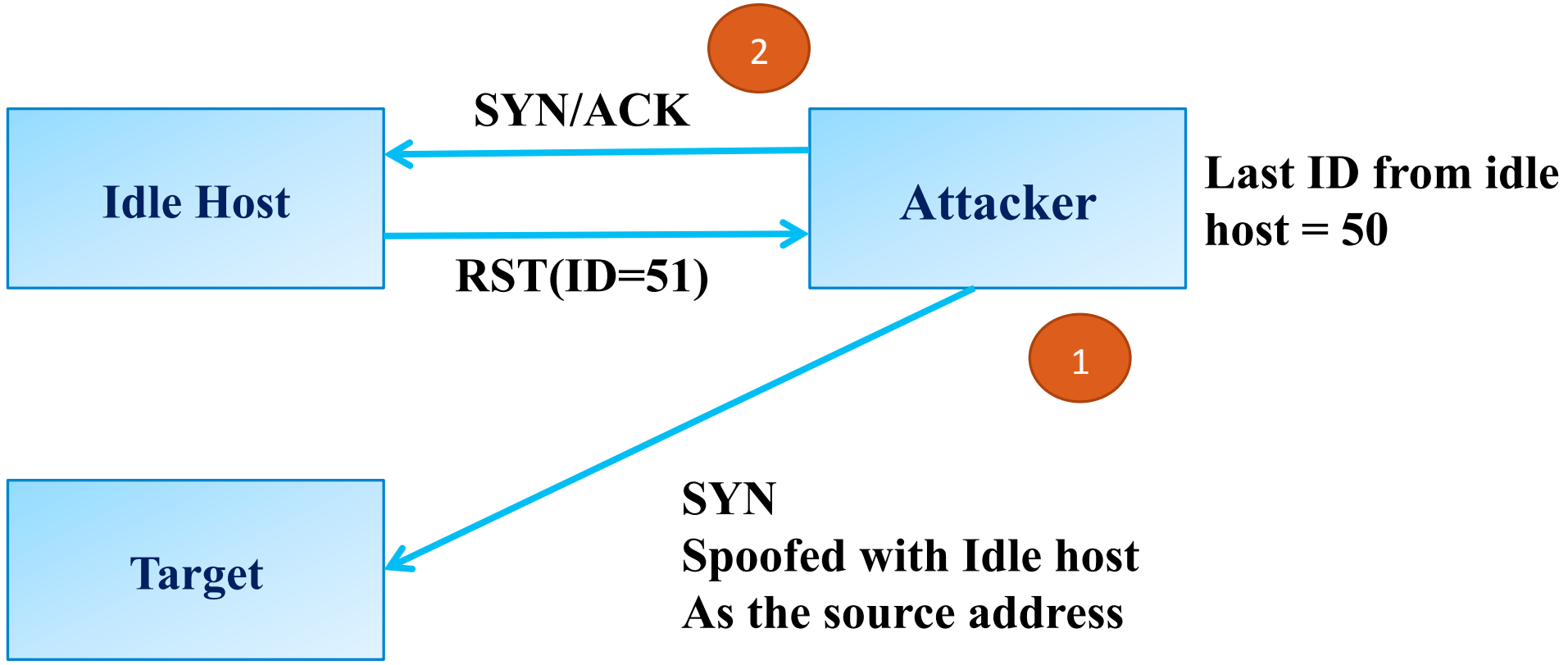
- The attacker needs to find a usable idle host that :
  - Is not sending/receiving any network traffic
  - Has a TCP implementation that produces predictable IP IDs that change by a known and fixed amount increment with each packet.

- Idle scanning takes advantage of the misconception that predictable IP IDs are not security threats.

Idle Scanning: Port open on target



Idle Scanning: Port closed on target



Idle Scanning Steps

Step 1. Attacker gets the current IP ID of the idle host by contacting it with a SYN packet and observing the IP ID of the response.

- By repeating this process, the increment applied to the IP ID with each packet can be determined.

Step 2. An attacker sends a spoofed SYN packet with the idle host's IP address to a port on the target machine. One of the two things will happen:

- If the port is listening, a SYN/ACK packet will be sent back to the idle host. But since the idle host didn't actually send out the initial SYN packet, this response appears to be unsolicited to the idle host, and it responds by sending back an RST packet.
- If the port is not listening, the target machine does not send a SYN/ACK packet back to the idle host, so the idle host doesn't respond.

Idle Scanning Steps

Step 3. The attacker contacts the idle host again to determine how much the IP ID has been incremented.

- If incremented by one, the port on target machine is closed.
- If incremented by two, the port on the target machine is open.

- If this technique is used properly on an idle host that **doesn't have any logging capabilities**, the attacker can scan any target without even **revealing his/her IP addresses**.

Idle Scanning using nmap

- After finding a suitable idle host, this type of scanning can be done by nmapusing –sI command-line option followed by idle host name.

```
reader@hacking:~/booksrc $ sudo nmap -sI  idlehost.com 192.168. 42.7
```

Reactive Defense vs Proactive Defense

Reactive Defense

- Intrusion detection systems detect port scans after information is leaked

Proactive Defense

- Prevent port scans before they actually happen.
- Is it possible ?

Proactive Defense against FIN, NULL and X-mas

- FUN, Null and X-mas rely on reset (RST) packets.

- If kernel never sends reset packets, FIN, NULL and X-mas scans will turn up nothing.
  - All we need to do is a simple kernel modification.

Proactive Defense against FIN, NULL and X-mas

Modifying kernel so that it never sends reset packets

Step 1: use grep to find kernel code responsible for sending reset packets.

```
reader@hacking:~/booksrc $ grep -n -A 20 “void.*send_reset” /usr/src/linux/net/ipv4/tcp_ipv4.c
```

Proactive Defense against FIN, NULL and X-mas

Modifying kernel so that it never sends reset packets

Step 2: Add a return command before the /* Never send a response to a reset*/ line.

Step 3: Recompile the kernel.

The resulting kernel won't sent out reset packets.

```
static void tcp_v4_send_reset(struct sock *sk, struct sk_buff *skb)
{
        struct tcphdr *th = tcp_hdr(skb);
        struct {
                struct tcphdr th;
#ifdef CONFIG_TCP_MD5SIG
                __be32 opt[(TCPOLEN_MD5SIG_ALIGNED >> 2)];
#endif
        } rep;
        struct ip_reply_arg arg;
#ifdef CONFIG_TCP_MD5SIG
        struct tcp_md5sig_key *key;
#endif
        struct net *net;

        /* Never send a reset in response to a reset. */
        if (th->rst)
                return;

        if (skb_rtable(skb)->rt_type != RTN_LOCAL)
                return;
```

Proactive Defense (shroud.c)

- Preventing information leakage with SYN and full-connect scans is a bit more difficult since open ports have to respond to SYN/ACK packets.

The idea behind Shroud.C

- If all of close ports also respond with SYN/ACK packets, the amount of useful information the attacker could retrieve will be minimized.
- In shroud.c the callback function spoofs a legitimate looking looking SYN/ACK response to any SYN packet. This will flood port scanners with a sea of false positive.

Shroud.C Source Code I

```
1  #include <libnet.h>
2  #include <pcap.h>
3  #include "hacking.h"
4
5  #define MAX_EXISTING_PORTS 30
6
7  void caught_packet(u_char *, const struct pcap_pkthdr *, const u_char *);
8  int set_packet_filter(pcap_t *, struct in_addr *, u_short *);
9
10 struct data_pass {
11    int libnet_handle;
12    u_char *packet;
13 };
14
15 int main(int argc, char *argv[]) {
16    struct pcap_pkthdr cap_header;
17    const u_char *packet, *pkt_data;
18    pcap_t *pcap_handle;
19    char errbuf[PCAP_ERRBUF_SIZE]; // same size as LIBNET_ERRBUF_SIZE
20    char *device;
21    u_long target_ip;
22    int network, i;
23    struct data_pass critical_libnet_data;
24    u_short existing_ports[MAX_EXISTING_PORTS];
25
```

Shroud.C Source Code II

```
26      if((argc < 2) || (argc > MAX_EXISTING_PORTS+2)) {
27         if(argc > 2)
28            printf("Limited to tracking %d existing ports.\n", MAX_EXISTING_PORTS);
29         else
30            printf("Usage: %s <IP to shroud> [existing ports...]\n", argv[0]);
31         exit(0);
32      }
33
34      target_ip = libnet_name_resolve(argv[1], LIBNET_RESOLVE);
35      if (target_ip == -1)
36         fatal("Invalid target address");
37
38      for(i=2; i < argc; i++)
39         existing_ports[i-2] = (u_short) atoi(argv[i]);
40
41      existing_ports[argc-2] = 0;
42
43      device = pcap_lookupdev(errbuf);
44      if(device == NULL)
45         fatal(errbuf);
46
47      pcap_handle = pcap_open_live(device, 128, 1, 0, errbuf);
48      if(pcap_handle == NULL)
49         fatal(errbuf);
50
```

Shroud.C Source Code III

```
51      critical_libnet_data.libnet_handle = libnet_open_raw_sock(IPPROTO_RAW);
52      if(critical_libnet_data.libnet_handle == -1)
53         libnet_error(LIBNET_ERR_FATAL, "can't open network interface.  -- this program must run as root.\n");
54
55      libnet_init_packet(LIBNET_IP_H + LIBNET_TCP_H, &(critical_libnet_data.packet));
56      if (critical_libnet_data.packet == NULL)
57         libnet_error(LIBNET_ERR_FATAL, "can't initialize packet memory.\n");
58
59      libnet_seed_prand();
60
61      set_packet_filter(pcap_handle, (struct in_addr *)&target_ip, existing_ports);
62
63      pcap_loop(pcap_handle, -1, caught_packet, (u_char *)&critical_libnet_data);
64      pcap_close(pcap_handle);
65   }
66
67   /* sets a packet filter to look for established TCP connections to target_ip */
68   int set_packet_filter(pcap_t *pcap_hdl, struct in_addr *target_ip, u_short *ports) {
69      struct bpf_program filter;
70      char *str_ptr, filter_string[90 + (25 * MAX_EXISTING_PORTS)];
71      int i=0;
72
73      sprintf(filter_string, "dst host %s and ", inet_ntoa(*target_ip)); // target IP
74      strcat(filter_string, "tcp[tcpflags] & tcp-syn != 0 and tcp[tcpflags] & tcp-ack = 0");
75
```

Shroud.C Source Code IV

```
76      if(ports[0] != 0) { // if there is at least one existing port
77         str_ptr = filter_string + strlen(filter_string);
78         if(ports[1] == 0) // there is only one existing port
79            sprintf(str_ptr, " and not dst port %hu", ports[i]);
80         else { // two or more existing ports
81            sprintf(str_ptr, " and not (dst port %hu", ports[i++]);
82            while(ports[i] != 0) {
83               str_ptr = filter_string + strlen(filter_string);
84               sprintf(str_ptr, " or dst port %hu", ports[i++]);
85            }
86            strcat(filter_string, ")");
87         }
88      }
89      printf("DEBUG: filter string is \'%s\'\n", filter_string);
90      if(pcap_compile(pcap_hdl, &filter, filter_string, 0, 0) == -1)
91         fatal("pcap_compile failed");
92
93      if(pcap_setfilter(pcap_hdl, &filter) == -1)
94         fatal("pcap_setfilter failed");
95   }
96
97   void caught_packet(u_char *user_args, const struct pcap_pkthdr *cap_header, const u_char *packet) {
98      u_char *pkt_data;
99      struct libnet_ip_hdr *IPhdr;
100     struct libnet_tcp_hdr *TCPhdr;
```

Shroud.C Source Code V

```
101     struct data_pass *passed;
102     int bcount;
103
104     passed = (struct data_pass *) user_args; // pass data using a pointer to a struct
105
106     IPhdr = (struct libnet_ip_hdr *) (packet + LIBNET_ETH_H);
107     TCPhdr = (struct libnet_tcp_hdr *) (packet + LIBNET_ETH_H + LIBNET_TCP_H);
108
109     libnet_build_ip(LIBNET_TCP_H,      // size of the packet sans IP header
110        IPTOS_LOWDELAY,                 // IP tos
111        libnet_get_prand(LIBNET_PRu16), // IP ID (randomized)
112        0,                              // frag stuff
113        libnet_get_prand(LIBNET_PR8),   // TTL (randomized)
114        IPPROTO_TCP,                    // transport protocol
115        *((u_long *)&(IPhdr->ip_dst)),  // source IP (pretend we are dst)
116        *((u_long *)&(IPhdr->ip_src)),  // destination IP (send back to src)
117        NULL,                           // payload (none)
118        0,                              // payload length
119        passed->packet);                // packet header memory
120
121     libnet_build_tcp(htons(TCPhdr->th_dport),// source TCP port (pretend we are dst)
122        htons(TCPhdr->th_sport),        // destination TCP port (send back to src)
123        htonl(TCPhdr->th_ack),          // sequence number (use previous ack)
124        htonl((TCPhdr->th_seq) + 1),    // acknowledgement number (SYN's seq # + 1)
125        TH_SYN | TH_ACK,                // control flags (RST flag set only)
```

Shroud.C Source Code VI

```
126         libnet_get_prand(LIBNET_PRu16), // window size (randomized)
127         0,                              // urgent pointer
128         NULL,                           // payload (none)
129         0,                              // payload length
130         (passed->packet) + LIBNET_IP_H);// packet header memory
131
132     if (libnet_do_checksum(passed->packet, IPPROTO_TCP, LIBNET_TCP_H) == -1)
133         libnet_error(LIBNET_ERR_FATAL, "can't compute checksum\n");
134
135     bcount = libnet_write_ip(passed->libnet_handle, passed->packet, LIBNET_IP_H+LIBNET_TCP_H);
136     if (bcount < LIBNET_IP_H + LIBNET_TCP_H)
137         libnet_error(LIBNET_ERR_WARNING, "Warning: Incomplete packet written.");
138     printf("bing!\n");
139 }
```

Notes about Shroud.C

- When the program is compiled and executed, it will shroud the IP address given as the first argument, with the exception of a list of existing ports provided as the remaining arguments.

- While shroud is running, any port scanning attempts will show every port to be open.

Reading Materials

Textbook

- Hacking: The art of exploitation, Jon Erickson
  - Chapter 4 - From 0x470 to 0x480

Source Codes

- https://github.com/shichao-an/hacking/blob/master/shroud.c
- https://github.com/Hybridmax/G92XF_Mystery_Kernel_Old/blob/master/net/mptcp/mptcp_ipv4.c