数据包伪造编程

数据包伪造

- 伪造包头或包中信息
- 大量的网络攻击都与数据包伪造相关

正常的Socket发送数据

```
void main()
{
    struct sockaddr_in dest_info;
    char *data = "UDP message\n";

    // Step 1: Create a network socket
    int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    // Step 2: Provide information about destination.
    memset((char *) &dest_info, 0, sizeof(dest_info));
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr.s_addr = inet_addr("10.0.2.5");
    dest_info.sin_port = htons(9090);

    // Step 3: Send out the packet.
    sendto(sock, data, strlen(data), 0,
                  (struct sockaddr *)&dest_info, sizeof(dest_info));
    close(sock);
}
```

Testing: Use the `netcat` (`nc`) command to run a UDP server on `10.0.2.5`. We then run the program on the left from another machine. We can see that the message has been delivered to the server machine:

```
seed@Server(10.0.2.5):$ nc -luv 9090
Connection from 10.0.2.6 port 9090 [udp/*] accepted
UDP message
```

使用Raw Socket发送信息

- 数据包伪造的两个重要步骤:
  - 构造数据包
  - 发送数据包

数据包发送

```
/*****************************************************************
  Given an IP packet, send it out using a raw socket.
*****************************************************************/
void send_raw_ip_packet(struct ipheader* ip)
{
    struct sockaddr_in dest_info;
    int enable = 1;

    // Step 1: Create a raw network socket.
    int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

    // Step 2: Set socket option.
    setsockopt(sock, IPPROTO_IP, IP_HDRINCL,
                     &enable, sizeof(enable));

    // Step 3: Provide needed information about destination.
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr = ip->iph_destip;

    // Step 4: Send the packet out.
    sendto(sock, ip, ntohs(ip->iph_len), 0,
           (struct sockaddr *)&dest_info, sizeof(dest_info));
    close(sock);
}
```

We use *setsockopt()* to enable *IP_HDRINCL* on the socket.

For raw socket programming, since the destination information is already included in the provided IP header, we do not need to fill all the fields

Since the socket type is raw socket, the system will send out the IP packet as is.

数据包构造：ICMP

Fill in the ICMP Header

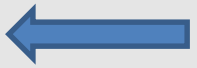
```
char buffer[1500];

memset(buffer, 0, 1500);

/*****************************************************
   Step 1: Fill in the ICMP header.
 *****************************************************/
struct icmpheader *icmp = (struct icmpheader *)
                          (buffer + sizeof(struct ipheader));
icmp->icmp_type = 8; //ICMP Type: 8 is request, 0 is reply.

// Calculate the checksum for integrity
icmp->icmp_chksum = 0;
icmp->icmp_chksum = in_cksum((unsigned short *)icmp,
                             sizeof(struct icmpheader));
```

Find the starting point of the ICMP header, and typecast it to the ICMP structure

Fill in the ICMP header fields

Fill in the IP Header

```
/*****************************************************
   Step 2: Fill in the IP header.
 *****************************************************/
struct ipheader *ip = (struct ipheader *) buffer;
ip->iph_ver = 4;
ip->iph_ihl = 5;
ip->iph_ttl = 20;
ip->iph_sourceip.s_addr = inet_addr("1.2.3.4");
ip->iph_destip.s_addr = inet_addr("10.0.2.5");
ip->iph_protocol = IPPROTO_ICMP;
ip->iph_len = htons(sizeof(struct ipheader) +
                    sizeof(struct icmpheader));
```



Typecast the buffer to the IP structure

Fill in the IP header fields

Finally, send out the packet

```
send_raw_ip_packet (ip);
```

数据包构造：UDP

```
memset(buffer, 0, 1500);
struct ipheader *ip = (struct ipheader *) buffer;
struct udpheader *udp = (struct udpheader *) (buffer +
                                    sizeof(struct ipheader));

/*********************************************************
   Step 1: Fill in the UDP data field.
 *********************************************************/
char *data = buffer + sizeof(struct ipheader) +
                      sizeof(struct udpheader);
const char *msg = "Hello Server!\n";
int data_len = strlen(msg);
strncpy (data, msg, data_len);

/*********************************************************
   Step 2: Fill in the UDP header.
 *********************************************************/
udp->udp_sport = htons(12345);
udp->udp_dport = htons(9090);
udp->udp_ulen = htons(sizeof(struct udpheader) + data_len);
udp->udp_sum =  0; /* Many OSes ignore this field, so we do not
                      calculate it. */
```

- Constructing UDP packets is similar, except that we need to include the payload data now.

```
/*****************************************************
   Step 3: Fill in the IP header.
 *****************************************************/

...... /* Code omitted here; same as that in Listing 12.6 */
ip->iph_protocol = IPPROTO_UDP; // The value is 17.
ip->iph_len = htons(sizeof(struct ipheader) +
                    sizeof(struct udpheader) + data_len);
```

Testing: Use the `nc` command to run a UDP server on `10.0.2.5`. We then spoof a UDP packet from another machine. We can see that the spoofed UDP packet was received by the server machine.

```
seed@Server(10.0.2.5):$ nc -luv 9090
Connection from 1.2.3.4 port 9090 [udp/*] accepted
Hello Server!
```

计算校验和

```
unsigned short in_cksum (unsigned short *buf, int length)
{
   unsigned short *w = buf;
   int nleft = length;
   int sum = 0;
   unsigned short temp=0;

   /*
    * The algorithm uses a 32 bit accumulator (sum), adds
    * sequential 16 bit words to it, and at the end, folds back all
    * the carry bits from the top 16 bits into the lower 16 bits.
    */
   while (nleft > 1)  {
       sum += *w++;
       nleft -= 2;
   }

   /* treat the odd byte at the end, if any */
   if (nleft == 1) {
        *(u_char *)(&temp) = *(u_char *)w ;
        sum += temp;
   }

   /* add back carry outs from top 16 bits to low 16 bits */
   sum = (sum >> 16) + (sum & 0xffff);  // add hi 16 to low 16
   sum += (sum >> 16);                  // add carry
   return (unsigned short)(~sum);
}
```

- RFC 1071中描述了IP、ICMP、TCP、UDP的数据包头中校验和计算方法。

- 计算步骤：
  - 将校验和字段置为0。
  - 将每两个字节（16位）相加（二进制求和）直到最后得出结果。若最后还剩一个字节，扩展成16位，继续与前面结果相加。
  - 将高16位与低16位相加，直到高16位为0为止。
  - 将最后的结果（二进制）取反。

Pcap?

- Winpcap
- Libnet
- Scapy
- Netwox

进一步：先嗅探再伪造

- 很多场合下，我们需要先监听到数据包，再对它进行伪造。
- 步骤 (using UDP as example)
  - Use PCAP API to capture the packets of interests
  - Make a copy from the captured packet
  - Replace the UDP data field with a new message and swap the source and destination fields
  - Send out the spoofed reply

UDP Packet

```
void spoof_reply(struct ipheader* ip)
{
    const char buffer[1500];
    int ip_header_len = ip->iph_ihl * 4;
    struct udpheader* udp = (struct udpheader *) ((u_char *)ip +
                                                  ip_header_len);
   if (ntohs(udp->udp_dport) != 9999) {
       // Only spoof UDP packet with destination port 9999
       return;
   }

   // Step 1: Make a copy from the original packet
   memset((char*)buffer, 0, 1500);
   memcpy((char*)buffer, ip, ntohs(ip->iph_len));
   struct ipheader  * newip  = (struct ipheader *) buffer;
   struct udpheader * newudp = (struct udpheader *) (buffer +
   ip_header_len);
   char *data = (char *)newudp + sizeof(struct udpheader);

   // Step 2: Construct the UDP payload, keep track of payload size
   const char *msg = "This is a spoofed reply!\n";
   int data_len = strlen(msg);
   strncpy (data, msg, data_len);
```

```
    // Step 3: Construct the UDP Header
    newudp->udp_sport = udp->udp_dport;
    newudp->udp_dport = udp->udp_sport;
    newudp->udp_ulen = htons(sizeof(struct udpheader) + data_len);
    newudp->udp_sum =  0;

    // Step 4: Construct the IP header (no change for other fields)
    newip->iph_sourceip = ip->iph_destip;
    newip->iph_destip = ip->iph_sourceip;
    newip->iph_ttl = 50; // Rest the TTL field
    newip->iph_len = htons(sizeof(struct ipheader) +
                           sizeof(struct udpheader) + data_len);

    // Step 5: Send out the spoofed IP packet
    send_raw_ip_packet(newip);
}
```