# A Topological Characterization of TCP/IP Security

Giovanni Vigna

Reliable Software Group
Department of Computer Science
University of California Santa Barbara
vigna@cs.ucsb.edu

**Abstract.** The TCP/IP protocol suite has been designed to provide a simple, open communication infrastructure in an academic, collaborative environment. Therefore, the TCP/IP protocols are not able to provide the authentication, integrity, and privacy mechanisms to protect communication in a hostile environment. To solve these security problems, a number of application-level protocols have been designed and implemented on top of TCP/IP. In addition, *ad hoc* techniques have been developed to protect networks from TCP/IP-based attacks. Nonetheless, a formal approach to TCP/IP security is still lacking. This work presents a formal model of TCP/IP networks and describes some well-known attacks using the model. The topological characterization of TCP/IP-based attacks enables better understanding of the vulnerabilities and supports the design of tougher detection, protection, and testing tools.
**Keywords:** Security, TCP/IP protocols, Network model, Topology.

## 1 Introduction

Security has become a major concern in the ever-growing Internet. Break-ins are becoming wide-spread and more sophisticated [6]. Some of these attacks either exploit or are based on some well-known flaws in the TCP/IP protocol suite [21, 4].

The TCP/IP protocol suite was designed more than twenty years ago [32] to provide a simple, efficient, open communication infrastructure in a collaborative and friendly environment. Little attention was paid to security issues, at that time. Therefore, the most used version of the TCP/IP protocol suite (namely, version 4) does not provide authentication, integrity, and privacy mechanisms to protect communication in a potentially hostile environment. To provide secure communication services, a number of secure higher-level protocols (e.g., Kerberos [17] and SSL [13]) have been designed and implemented on top of the TCP/IP stack. In order to protect TCP/IP networks from attacks based on those protocols *ad hoc* techniques (e.g., firewalls [8, 7]) have been developed. In addition, a secure version of the TCP/IP protocol suite (called version 6) has been designed. Nonetheless, the original (insecure) protocols are still widely used.

Even if in the last few years a more systematic approach to the problem has been followed, a formal approach to TCP/IP security is still lacking. This work presents a formal model of TCP/IP networks. The model is used to describe several well-known attacks that exploit vulnerabilities in the TCP/IP protocol

suite. The analysis is carried out from a topological viewpoint aimed at identifying the prerequisites to mount a particular attack and the conditions that enable both detection and protection.

The examples included in the paper show that the model supports a precise description of the attacks and provides effective support to network security analysis. The network model described in this paper has been successfully used as the basis for intrusion detection sensor placement [30].

The rest of the paper is structured as follows. In Section 2 we shortly present the TCP/IP protocol suite. In Section 3 we present our model of TCP/IP networks. In Sections 4, 5, and 6 we describe some of the vulnerabilities of the TCP/IP protocol suite using the proposed model. In Section 7 we discuss how the outcomes of the previous sections can be used to protect TCP/IP networks. In Section 8 we draw some conclusions and outline future work.

## 2 The TCP/IP Protocol Suite

The Internet is a collection of computer networks, which share the same network protocol suite, namely TCP/IP. Networks participating in the Internet may adopt any protocol at the link layer (e.g., Ethernet, Token Ring, etc.), but they must adopt the TCP/IP protocol suite in the upper layers, i.e., the *Internet Protocol* (IP) at the network level, and the *Transmission Control Protocol* (TCP) or the *User Datagram Protocol* (UDP) at the transport level[1]. Since the TCP/IP protocol suite does not include any session and presentation layer protocols, application-level protocols (e.g., SSH or FTP) are based directly on TCP/UDP protocols.

### 2.1 Internet Protocol

The IP protocol [25] provides an unreliable, best-effort, connectionless packet delivery service. Each possible source or destination of a message in the network (i.e., each network interface) has a unique IP address, composed of 32 bits (e.g., `192.168.5.10`). Hosts may have one or more network interfaces, and, therefore they may have more than one IP address.

An IP *datagram* is composed of a header and a payload. The header contains, among other fields, the source and destination IP addresses, the type of the payload and possible options. Every host in the Internet can communicate with every other host by sending an IP datagram containing the recipient IP address. The datagram is forwarded by a series of gateways until it reaches the intended destination.

The process of determining the path of a datagram in an internet is called *routing*. In the simplest case, both source and destination hosts reside in the same physical subnetwork, and, therefore, the datagram is delivered directly, encapsulating the IP datagram in a link-level message. However, if the source and destination hosts are in different subnetworks, the datagram must be delivered indirectly. In this case, the source host sends the datagram to a directly connected gateway. The gateway compares the destination IP address to its routing table and it decides if it can deliver the datagram directly or if it must send the datagram to another gateway. Routing tables entries are either static or defined dynamically by means of *routing protocols* (e.g., RIP [18], BGP [28]).

[1] We adopt the ISO/OSI reference model [12].

### 2.2 UDP

The User Datagram Protocol [24] relies on IP to provide an unreliable, best-effort, connectionless datagram delivery service. UDP datagrams are encapsulated in IP datagrams. A UDP datagram contains a source and destination port number and a payload. The port numbers are used to distinguish different datagram sources and destinations for a single IP address. UDP is used mostly for request/reply services (e.g., NFS [19]).

### 2.3 TCP

The Transmission Control Protocol [26] relies on IP to provide a reliable, best-effort, connection-oriented, full-duplex stream delivery service. The TCP protocol uses the port abstraction in order to distinguish among different virtual circuits between the same IP addresses. A virtual circuit is uniquely identified by the tuple *(source IP address, destination IP address, source TCP port, destination TCP port)*. TCP segments are encapsulated into IP datagrams. Each TCP segment has a header and a payload. The header contains the source and destination port numbers, which, together with the IP addresses of the enclosing IP datagram, identify the virtual circuit the segment belongs to. In addition, the TCP header contains a sequence number that identifies the position of the first byte of the payload with respect to the circuit's stream and an acknowledgment number that identifies the next byte that the segment source expects from the destination. Finally, the header contains a set of flags that are used during the setup and shutdown phase of the TCP virtual circuit.

The setup phase of a TCP virtual circuit is a three-way handshake between a client that performs an *active open* from port $p_c$ and a server that performs a *passive open* at port $p_s$. The three-way handshake is performed according to the following steps:

1. the client sends a segment to the server with the SYN flag set and containing an initial sequence number $s_c^0$;
2. the server replies with a segment containing its initial sequence number $s_s^0$ and the acknowledgment of the client sequence number $a_s^0 = s_c^0 + 1$; the segment is marked with both the SYN and ACK flags;
3. the client sends a segment with the ACK flag set, containing the sequence number $s_c^1 = s_c^0 + 1$ and the acknowledgment number $a_c^1 = s_s^0 + 1$.

From this point on, the session is in an *established* state, and each of the communication partners can send a segment that will be acknowledged by the other. When one of the partners decides to stop transmitting data, it sends a segment marked with the FIN flag. The other partner acknowledges the message and eventually produces a similar segment, shutting down the connection. At any moment, both partners may send a segment marked with the RST flag to immediately shut down the connection.

### 2.4 Application protocols

Application protocols can be implemented on top of either TCP or UDP, depending of the underlying requirements (e.g., reliability). For example, the Simple Mail Transfer Protocol (SMTP) [27] and the Hypertext Transfer Protocol

(HTTP) [11] are both implemented on top of TCP, while the Domain Name System [20] is implemented on top UDP[2].

### 2.5 Security issues

The TCP/IP protocol suite has been designed to deliver robust communication services in a cooperative, friendly network environment. As a consequence, the TCP/IP protocol suite is vulnerable to a number of attacks, such as spoofing, sniffing, and hijacking. In the following sections, we first introduce a model for networks and the TCP/IP protocol suite. Then, we use the model to describe some basic attacks against the TCP/IP protocol suite.

## 3 A Network Model

We model a *network* as a connected hypergraph [5] $N$ on the set of *interfaces* $I = \{i_1, i_2, \ldots, i_n\}$. More precisely, a network is a family $N = \{E_1, E_2, \ldots, E_m\}$ where each $E_i$ is a subset of $I$ and it is called an *edge* of the hypergraph. The following statements hold:

$$E_i \neq \emptyset \qquad (i = 1, 2, \ldots, m), \tag{1}$$
$$\bigcup_{i=1}^{m} E_i = I.$$

We define a *route* between two interfaces $i_j$ and $i_k$ as a sequence of edges $r = \langle E_{r_1}, E_{r_2}, \ldots, E_{r_n} \rangle$ with $i_j \in E_{r_1}$, $i_k \in E_{r_n}$, and, $E_{r_l} \cap E_{r_{l+1}} \neq \emptyset$ for $l < n$. We call $R$ the set of all routes.

We partition the set of edges $N$ in two parts: the set *hosts* $H = \{H_1, H_2, \ldots, H_p\}$ and the set *links* $L = \{L_1, L_2, \ldots, L_q\}$. Hosts are edges (with at least one interface by (1)) which partition the set of interfaces, i.e.:

$$\bigcup_{i=1}^{p} H_i = I$$
$$H_i \cap H_j = \emptyset \qquad i, j \in \{1, 2, \ldots, p\} \text{ and } i \neq j. \tag{2}$$

Links are edges containing at least two interfaces. They also partition the set $I$:

$$\min_j \mid L_j \mid \geq 2$$
$$\bigcup_{i=1}^{q} L_i = I$$
$$L_i \cap L_j = \emptyset \qquad i, j \in \{1, 2, \ldots, q\} \text{ and } i \neq j. \tag{3}$$

From the previous formulas it follows that any route in the network alternates hosts and links[3].

[2] The DNS protocol can actually operate on top of both UDP and TCP.

[3] If not so, two hosts or two links may have a non-empty intersection, which is against (2) and (3), respectively.

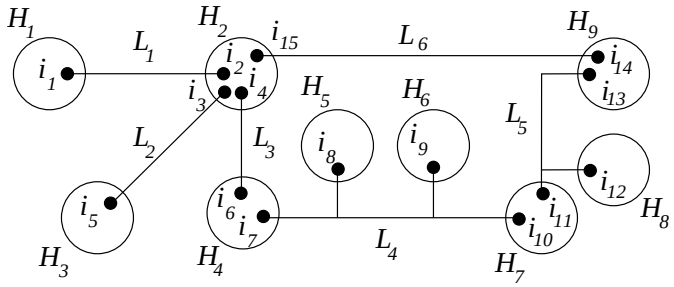



**Fig. 1.** A network hypergraph.

We define a graphic representation for the entities of the model. Hosts are drawn as circles. Interfaces belonging to a host are drawn as dots inside the corresponding circle. Links are drawn as lines connecting interfaces belonging to the link. The network topology may also be described by using the *hosts matrix HM* and the *links matrix LM*. The hosts matrix has columns representing the network hosts $H_1, H_2, \ldots, H_p$ and rows representing the interfaces $i_1, i_2, \ldots, i_n$. Element $e_k^j$ is 1 if $i_j \in H_k$ and 0 otherwise. The links matrix has columns representing the network links $L_1, L_2, \ldots, L_q$ and rows representing the interfaces $i_1, i_2, \ldots, i_n$. Element $e_k^j$ is 1 if $i_j \in L_k$ and zero otherwise. The juxtaposition of *HM* and *LM* gives the incidence matrix *IM* of the network hypergraph.

Figure 1 shows a network with nine hosts, six links, and fifteen interfaces:

$$
\begin{aligned}
I =& \{i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8, i_9, i_{10}, i_{11}, i_{12}, i_{13}, i_{14}, i_{15}\} \\
H =& \{\{i_1\}, \{i_2, i_3, i_4, i_{15}\}, \{i_5\}, \{i_6, i_7\}, \{i_8\}, \{i_9\}, \\
& \{i_{10}, i_{11}\}, \{i_{12}\}, \{i_{13}, i_{14}\}\} \\
L =& \{\{i_1, i_2\}, \{i_3, i_5\}, \{i_4, i_6\}, \{i_7, i_8, i_9, i_{10}\}, \{i_{11}, i_{12}, i_{13}\}, \{i_{14}, i_{15}\}\} \\
N =& \{H_1, \ldots, H_9, L_1, \ldots, L_6\}.
\end{aligned}
$$

The corresponding hosts and links matrices are shown in Figure 2.

| | $H_1$ | $H_2$ | $H_3$ | $H_4$ | $H_5$ | $H_6$ | $H_7$ | $H_8$ | $H_9$ |
|---|---|---|---|---|---|---|---|---|---|
| $i_1$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $i_2$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $i_3$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $i_4$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $i_5$ | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| $i_6$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| $i_7$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| $i_8$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| $i_9$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $i_{10}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| $i_{11}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| $i_{12}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $i_{13}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $i_{14}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $i_{15}$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | $L_1$ | $L_2$ | $L_3$ | $L_4$ | $L_5$ | $L_6$ |
|---|---|---|---|---|---|---|
| $i_1$ | 1 | 0 | 0 | 0 | 0 | 0 |
| $i_2$ | 1 | 0 | 0 | 0 | 0 | 0 |
| $i_3$ | 0 | 1 | 0 | 0 | 0 | 0 |
| $i_4$ | 0 | 0 | 1 | 0 | 0 | 0 |
| $i_5$ | 0 | 1 | 0 | 0 | 0 | 0 |
| $i_6$ | 0 | 0 | 1 | 0 | 0 | 0 |
| $i_7$ | 0 | 0 | 0 | 1 | 0 | 0 |
| $i_8$ | 0 | 0 | 0 | 1 | 0 | 0 |
| $i_9$ | 0 | 0 | 0 | 1 | 0 | 0 |
| $i_{10}$ | 0 | 0 | 0 | 1 | 0 | 0 |
| $i_{11}$ | 0 | 0 | 0 | 0 | 1 | 0 |
| $i_{12}$ | 0 | 0 | 0 | 0 | 1 | 0 |
| $i_{13}$ | 0 | 0 | 0 | 0 | 1 | 0 |
| $i_{14}$ | 0 | 0 | 0 | 0 | 0 | 1 |
| $i_{15}$ | 0 | 0 | 0 | 0 | 0 | 1 |

**Fig. 2.** Hosts and links matrices.

A host $H_i$ is *connected to* a link $L_j$ iff $H_i \cap L_j \neq \emptyset$. A link $L_j$ *connects* a set of hosts $C(L_j)$ with:

$$C(L_j) = \{H_i \in H \mid H_i \cap L_j \neq \emptyset\}.$$

Two hosts $H_i$, $H_j$ are said to be *connected by* a link $L_k$ iff they both belong to the set of hosts the link connects, i.e., $H_i \in C(L_k) \wedge H_j \in C(L_k)$.

A *subnetwork* is a sub-hypergraph composed of a non-empty set of links $L_J = \{L_i \in L \mid i \in J \text{ and } J \subseteq \{1, 2, \ldots, q\}\}$ together with the hosts they connect, i.e.:

$$N_J = L_J \cup \{H_i \in H \mid \exists j \in J \text{ such that } H_i \in C(L_j)\}.$$

A *message* $m \in M$ is a triple containing two interfaces $i_s$ (the *source* interface) and $i_d$ (the *destination* interface), and a payload $x \in X$ (for some set $X$, to be specified later) such that both interfaces belong to the same link, i.e.:

$$m = (i_s, i_d, x) \text{ such that } i_s \in L_i \text{ and } i_d \in L_i \text{ with } L_i \in L.$$

A message $m = (i_s, i_d, x)$ is said to be *between* hosts $H_i$ (the *source* host) and $H_j$ (the *destination* host), with $i_s \in H_i$ and $i_d \in H_j$, and it is said to *appear* on link $L_i$, if $i_s, i_d \in L_i$.

We introduce a set of functions that extract the source/destination interface/host, the payload, and the associated link from a message:

$$\begin{aligned}
&si : M \to I, \text{ defined by } si((i_s, i_d, x)) = i_s;\\
&sh : M \to H, \text{ defined by } sh((i_s, i_d, x)) = H_i, \text{ where } i_s \in H_i;\\
&di : M \to I, \text{ defined by } di((i_s, i_d, x)) = i_d;\\
&dh : M \to H, \text{ defined by } dh((i_s, i_d, x)) = H_j, \text{ where } i_d \in H_j;\\
&pl : M \to P, \text{ defined by } pl((i_s, i_d, x)) = x;\\
&al : M \to L, \text{ defined by } al((i_s, i_d, x)) = L_i, \text{ where } i_s, i_d \in L_i.
\end{aligned}$$

The use of hypergraphs to describe network topologies allows for intuitive modeling of networks based on shared bus technologies, e.g., Ethernet. In these technologies, hosts that are connected to a shared communication link may eavesdrop the messages exchanged using such link even if they are not directly involved in the communication.

Given a message $m$, we define the set of *listeners* of the message as the set of hosts connected to the link the message appears on, and owning neither of the interfaces used to send or receive the message:

$$ML(m) = \{H_i \in H \mid H_i \in C(al(m)) \ \wedge \ si(m), di(m) \notin H_i\}.$$

Current shared bus technologies provide mechanisms to prevent listeners to actively intercept traffic. For example, Ethernet switches propagate messages selectively to the intended destinations only. Unfortunately, the mechanisms implemented by these technologies can be easily bypassed (see for example [29]). Therefore, it is important to model shared bus technologies as hypergraphs to represent the fact that it is possible that, under certain conditions, a host not involved in the communication may access the messages. This is also true in

the case of wireless networks, where the communication medium is physically shared.

Note that it is possible for a node to connect to a link and passively eavesdrop the traffic without ever participating in any message exchange. These nodes will never generate a message and may not even have an associated IP address. Therefore, such nodes are impossible to spot or detect reliably [22]. Nodes of this type are not taken into account in our model.

### 3.1 IP Networks

An IP network is a network with a injective function $addr : I \to A$ which is a mapping between IP addresses $A = \{a_1, a_2, \ldots, a_n\}$ and interfaces.

IP datagrams are exchanged between endpoints using a number of link-level messages. Therefore, we model an IP datagram $d \in D$, as a sequence of messages $\langle m_1, m_2, \ldots, m_n \rangle$ such that:

1. the payload $x$ of each message $m_i$ is a triple $(a_s, a_d, x')$, where $a_s \in A$ is the source address, $a_d \in A$ is the destination address and $x' \in X'$ (to be specified later) is the datagram payload. A datagram is said to be *between* hosts $H_i$ (the source host) and $H_j$ (the destination host), with $addr^{-1}(a_s) \in H_i$ and $addr^{-1}(a_d) \in H_j$. We introduce a set of functions to extract the source/destination addresses/hosts and the payload from a message $m_i$ belonging to a datagram $d$:

   $dsa : M \to A$, defined by $dsa((i_{s_i}, i_{d_i}, (a_s, a_d, x'))) = a_s$;
   $dsh : M \to H$, defined by
   $dsh((i_{s_i}, i_{d_i}, (a_s, a_d, x'))) = H_i \in H$, where $addr^{-1}(a_s) \in H_i$;
   $dda : M \to A$, defined by $dda((i_{s_i}, i_{d_i}, (a_s, a_d, x'))) = a_d$;
   $ddh : M \to H$, defined by
   $ddh((i_{s_i}, i_{d_i}, (a_s, a_d, x'))) = H_j \in H$, where $addr^{-1}(a_d) \in H_j$;
   $dpl : M \to P'$ defined by $dpl((i_{s_i}, i_{d_i}, (a_s, a_d, x'))) = x'$.

2. Given two consecutive messages $m_j$ and $m_{j+1}$ in a datagram $d$, the destination interface of $m_j$ and the source interface of $m_{j+1}$ belong to the same host $H_f$:

   $$di(m_j) \in H_f \wedge si(m_{j+1}) \in H_f.$$

   $H_f$ is said to be a *forwarder* of $d$. We define the partial function *forw* that given two messages $m_i$ and $m_j$ returns the forwarder host if it exists, and otherwise returns $\perp$, which means "undefined":

   $forw : M \times M \to H \cup \{\perp\}$, defined by

   $$forw((i_{s_i}, i_{d_i}, x_i), (i_{s_j}, i_{d_j}, x_j)) = \begin{cases} H_f \text{ if } i_{d_i} \in H_f \wedge i_{s_j} \in H_f \\ \perp \quad \text{otherwise.} \end{cases}$$

   In a datagram, the sequence of links and hosts generated analyzing the sequence of messages, must represent a route in the network. More precisely,

the following sequence:

$$\begin{aligned}&\langle L_{l_1} = al(m_1), H_{h_1} = forw(m_1, m_2), \ldots,\\ &\ L_{l_i} = al(m_i), H_{h_i} = forw(m_i, m_{i+1}), L_{l_{i+1}} = al(m_{i+1}), \ldots,\\ &\ L_{l_n} = al(m_n)\rangle\end{aligned} \tag{4}$$

is defined for each couple $(m_i, m_{i+1})$, has no repeated elements, and represents a route in the network, which is called the *datagram route*. We define a function *route* that given a datagram $d$ returns its route:

$$\begin{aligned}&route : D \to E^+, \text{ defined by}\\ &\qquad route(\langle m_1, m_2, \ldots, m_n\rangle) = \langle L_{l_1}, H_{h_1}, \ldots, H_{h_{n-1}}, L_{l_n}\rangle\\ &\qquad \text{as defined by (4).}\end{aligned}$$

A datagram $\langle m_1, m_2, \ldots, m_n\rangle$ is *well-formed* if the source interface of the first message and the destination interface of the last message are mapped respectively into the source and destination addresses contained in the message payload, i.e.:

$$addr(si(m_1)) = dsa(m_i) \text{ and } addr(di(m_n)) = dda(m_i).$$

For a given datagram $d$, we define the *listeners* of the datagram as the union of the listeners of every message of the datagram:

$$DL(\langle m_1, \ldots, m_n\rangle) = \bigcup_{j=1}^{n} ML(m_j).$$

In addition, we define the set of *forwarders* of a datagram $d$ as the set of hosts that appear in the datagram route:

$$DF(d) = \{H_i \mid H_i \in route(d)\}.$$

We define a *session* $s$ as a sequence of IP datagrams $\langle d_1, d_2, \ldots, d_n\rangle$ that represent a conversation between two IP addresses. Therefore, for each $d_i \in s$ with $i \neq n$, the following holds:

$$\forall m \in d_i, \forall m' \in d_{i+1}\ \{dsa(m), dda(m)\} = \{dsa(m'), dda(m')\}.$$

Note that datagrams belonging to the same session may follow different routes.

### 3.2 UDP Interactions

The UDP protocol provides an unreliable datagram delivery service over IP (see Section 2.2). A *UDP datagram* $u$ is an IP datagram $d$ whose (IP) payload $x'$ is a tuple $(p_s, p_d, x'')$ where $p_s, p_d \in \mathbb{N}$ represent the source and destination port respectively, and $x'' \in X''$ represents the (UDP) datagram payload.

There is no concept of a UDP session, but since a number of application protocols based on UDP (like DNS or NFS) follow a query/reply schema, we define a *UDP interaction* as an IP session composed of two UDP datagrams $\langle u^1, u^2\rangle$, whose payloads are respectively: $(p_s, p_d, x''^1)$ and $(p_d, p_s, x''^2)$.

For example, a UDP interaction between the two endpoints specified by addresses/ports $(a_i, p_i)$ and $(a_j, p_j)$ is composed of two datagrams $d^1 = \langle m_1^1, m_2^1, \ldots, m_p^1 \rangle$ and $d^2 = \langle m_1^2, m_2^2, \ldots, m_q^2 \rangle$ such that:

$$pl(m_i^1) = (a_i, a_j, (p_i, p_j, x_i''^1)), \text{ and } pl(m_i^2) = (a_j, a_i, (p_j, p_i, x_j''^2)).$$

Note that UDP datagrams may be used outside UDP interactions, e.g., in multicast-based applications and SNMP traps. These datagrams can be easily modeled using the proposed formalism but are not described in details, for the sake of brevity.

### 3.3 TCP Sessions

The TCP protocol (see Section 2.3) provides a reliable stream delivery service, using the IP datagram delivery service. A *TCP segment* $s$ is an IP datagram $d$ whose payload $x'$ is a tuple $(p_s, p_d, seq, ack, F, x'')$ where $p_s, p_d \in \mathbb{N}$ represent the source and destination ports, $seq, ack \in \mathbb{N}$ are the sequence and acknowledgment numbers, $F$ is a subset of the set of *flags* $FLAGS = \{\mathsf{SYN}, \mathsf{ACK}, \mathsf{FIN}, \mathsf{RST}, \mathsf{PSH}, \mathsf{URG}\}$ representing the segment properties, and $x'' \in X''$ is the segment payload. The elements of $X''$ are unstructured sequences of bytes. Thus, the payload sets $X$ and $X'$ are defined by:

$$X = A \times A \times X'$$
$$X' = \mathbb{N} \times \mathbb{N} \times X'' \cup \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \wp(FLAGS) \times X''.$$

We define a *TCP session* between port $p_c$ at address $a_c$ of host $H_c$ and port $p_s$ at address $a_s$ of host $H_s$ as an IP session such that:

- the first three segments represent the connection setup handshake, which defines the initial sequence numbers:

$$s^1 = (a_c, a_s, (p_c, p_s, seq_c^1, 0, \{\mathsf{SYN}\}, x''_{empty}))$$
$$s^2 = (a_s, a_c, (p_s, p_c, seq_s^2, seq_c^1 + 1, \{\mathsf{SYN}, \mathsf{ACK}\}, x''_{empty}))$$
$$s^3 = (a_c, a_s, (p_c, p_s, seq_c^1 + 1, seq_s^2 + 1, \{\mathsf{ACK}\}, x''_{empty})),$$

  where $x''_{empty}$ represents a null payload;
- the subsequent segments are such that, having defined the function $size : P'' \to \mathbb{N}$ that given a TCP payload returns a natural number representing its size, we have:
  - given two consecutive segments:

$$s^i = (a_t, a_u, (p_t, p_u, seq_t^i, ack_t^i, flags_t^i, x_t''^i)), \text{ and}$$

$$s^{i+1} = (a_u, a_t, (p_u, p_t, seq_u^{i+1}, ack_u^{i+1}, flags_u^{i+1}, x_u''^{i+1})),$$

    the following holds:

$$seq_u^{i+1} \geq ack_t^i, \text{ and } ack_u^{i+1} \leq seq_t^i + size(x_t''^i).$$

In addition, if it is:

$$seq_u^{i+1} = ack_t^i, x_t''^{i+1} = x''_{empty}, ack_u^{i+1} = seq_t^i + size(x_t''^i), \text{ and}$$
$$\mathsf{ACK} \in flags_u^{i+1},$$

the session is in a "quiet" state, i.e., all the data exchanged between the communication partners have been acknowledged;

- given a segment:

$$s^i = (a_t, a_u, (p_t, p_u, seq_t^i, ack_t^i, flags_t^i, x_t''^i)),$$

the following segment of the session featuring the same source/destination addresses, i.e.:

$$s^{i+k} = (a_t, a_u, (p_t, p_u, seq_t^{i+k}, ack_t^{i+k}, flags_t^{i+k}, x_t''^{i+k})),$$

must be such that:

$$seq_t^{i+k} = seq_t^i + size(x_t''^i), \text{ and } ack_t^{i+k} \geq ack_t^i;$$

– after a pair of consecutive segments:

$$s^f = (a_t, a_u, (p_t, p_u, seq_t^f, ack_t^f, \{\mathsf{FIN}\}, x''_{empty})), \text{ and}$$

$$s^{f+1} = (a_u, a_t, (p_u, p_t, ack_t^f, seq_t^f + 1, \{\mathsf{ACK}\}, x''_{empty})),$$

representing the shutdown of one direction of the connection, every following segment from $a_t$ to $a_u$ is just an acknowledgment of a previous segment from $a_u$ to $a_t$ and it has a null payload. This goes on until the active direction of the connection is shut down with a similar message exchange;

– after a segment with a RST flag, no other segment exists.

This is a simplified model of the TCP/IP protocol suite. We do not model fragmentation of IP datagrams, IP options, several TCP mechanisms like retransmission, windows, etc. We modeled just the features that are useful to describe the attacks that are the subjects of the next sections.

## 4 Sniffing

Sniffing is one of the most exploited techniques used in TCP/IP networks to obtain sensitive information about hosts and eventually breach their security.

Technically, sniffing consists in eavesdropping network traffic by using hosts connected to shared-bus networks or by placing sniffer programs on Internet gateways. In the first case, the attacker sets the network interface of its host in *promiscuous mode* reading all link-level messages that are transmitted on that link, regardless of their intended destination. In the second case, the attacker installs a program on the gateway so that every packet that the gateway forwards is logged to a file. In both cases, the sniffer program looks for IP datagrams containing sensitive information that are transmitted in unencrypted form and logs such information. A typical example is represented by Telnet connections. Sniffers wait for TCP packets directed to port 23 with the SYN bit set, which

represent the request for the establishment of a new remote session. After the initial handshake and the presentation of the login banner, the connecting user must insert a user name and a password. Such information is transmitted in unencrypted form to the server host. The sniffer program extracts the characters inserted by the user from the segments transmitted from the client to the server host and logs the inserted user name and password. Eventually, the attacker is able to access the server host providing the stolen user name/password combination. In addition, the sniffer program is able to log the whole session, which could include sessions to other hosts and so on. In general, sniffing is used as a building block to mount more complex attacks.

In our model, given a datagram $d$ between two hosts $H_i$ and $H_j$, every host $H_k$ which is a listener or a forwarder of $d$ may mount a sniffing attack. Therefore, called $DS(d)$ the set of hosts that may sniff a datagram, we have:

$$DS(d) = DL(d) \cup DF(d).$$

Given a session $s = \langle d^1, d^2, \ldots, d^n \rangle$ the hosts that may access the whole information are:

$$SS(s) = \bigcap_{j=1}^{n} DS(d^j)$$

For example, in Figure 1, given the datagram

$$d = \langle (i_7, i_{10}, (a_7, a_{13}, p')), (i_{11}, i_{13}, (a_7, a_{13}, p')) \rangle$$

that represents a datagram from host $H_4$ to host $H_9$ along route $\langle L_4, H_7, L_5 \rangle$, it is: $SS(d) = \{H_5, H_6, H_7, H_8\}$.

## 5 Spoofing

When a host tries to impersonate another host in a communication, we are in presence of *spoofing* [9]. The impersonated host may have a privileged access to the attacked host or may be regarded as a source of sensitive information. For example, the *rsh* and *rlogin* protocols allow sessions to be established between hosts without the need to provide a password if the connecting host is *trusted*. This means that the name (or address) of the host is used as the only means of authentication. This is equivalent to induce the identity of the sender of a letter from the return address printed on the envelope rather then from the signature of the message.

### 5.1 Spoofing IP

At the IP level, a host $H_i$ may impersonate another host[4] $H_j$ toward a third host $H_k$ by producing a *spoofed datagram* $d$. Such datagram contains, as its source address, the address associated to one of the interfaces of $H_j$ even if the

[4] Actually spoofing is about impersonating *addresses* rather than *hosts*, but we will extend the expression in order to match the common use of the term.

first message of the datagram is generated from one of the interfaces of $H_i$, i.e., $d = \langle m_1, m_2, \ldots, m_n \rangle$ is spoofed if:

$$\forall m_i \in d,\ dsa(m_i) = a_s \text{ where } addr^{-1}(a_s) \in H_j, \text{ and } si(m_1) = i_s \in H_i \neq H_j$$

$H_k$ receives just message $m_n$ and considers the enclosed IP datagram as sent by $H_j$.

### 5.2 Spoofing UDP

The UDP protocol just adds the port concept to IP datagrams, and therefore spoofing single UDP datagrams is exactly as spoofing IP datagrams. But, since UDP is often used in request/reply protocols, it is useful to analyze how spoofing can be used to mount an attack in those cases. An host $H_i$ may spoof a host $H_j$ toward a third host $H_k$ in a UDP interaction $\langle u^1, u^2 \rangle$ by creating a bogus request or a bogus reply. The latter case, also called *session hijacking*, will be described in Section 6.

In the former case, host $H_i$ sends a spoofed UDP datagram $u^1$ in an attempt to access resources on the attacked host $H_k$ available to the impersonated host $H_j$. If the spoofing host is not interested in the reply payload, it may perform plain IP spoofing. Differently, if the reply contains information that is the aim of the spoofing attack, the spoofing host must be a sniffer of the reply datagram, i.e., $H_i \in DS(u^2)$. In addition, if the attacker wants to avoid the reply packet to reach the spoofed host (e.g., to avoid detection), it must be a forwarder of the *hypothetical* reply datagram. Thus, considering the hypothetical reply datagram $u_h^2 = \langle m_1, \ldots, m_n \rangle$ between $H_k$ and $H_j$ it must be $H_i \in DF(u_h^2)$. The actual datagram will be: $u^2 = \langle m_1, \ldots, m_j \rangle$ with $j < n$ and such that $di(m_j) \in H_i$, i.e., the datagram is intercepted by $H_i$ and it is not forwarded to $H_j$. In actual TCP/IP networks, attacks to routing mechanisms could be used in order to create this condition. Alternatively, the attacking host must make sure that the impersonated host is unable to process the reply datagram by waiting for the host being down or malfunctioning or by flooding the host with so much traffic that the datagram is dropped.

A protocol that is vulnerable to this attack is the NFS protocol [19]. NFS servers accept requests to access the exported file systems only from a restricted set of addresses. An attacker may use spoofing in order to impersonate an authorized address and therefore gain access to the exported file systems. In this case, if the attacker aims at obtaining some file contents, then it must be a sniffer of the reply. If the attack is aimed at executing some command (e.g., *delete* a file), and the attacker is not interested in the command results, then plain spoofing can be performed.

### 5.3 Spoofing TCP

A host $H_i$ may try to impersonate another host $H_j$ toward a third host $H_k$ not just at the datagram level but for a whole TCP session. This attack has been discussed in [4, 21].

In order to successfully impersonate host $H_j$, $H_i$ must be able to:

1. avoid that any segment sent by $H_k$ during the session reaches the spoofed host $H_j$;

2. determine the first sequence number produced by the attacked host $H_k$.

The first problem arises because if replies to spoofed segments are delivered to $H_j$, $H_j$ will induce that some error occurred and generate a reset segment directed to $H_k$, resulting in connection shutdown. This implies that $H_i$ must be an "intended" forwarder of every segment of the session, i.e., given the set of hypothetical segments $\{s_h^1, \ldots, s_h^n\}$ from $H_k$ to $H_j$, we have:

$$H_i \in \bigcap_{j=1}^{n} DF(s_h^j). \tag{5}$$

This means that all the segments, under normal circumstances, would follow a route that includes the attacking host. Alternatively, host $H_i$ must make host $H_j$ unable to process the reply segments. As said before, in an actual TCP/IP network there are a number of ways to achieve this effect.

The second condition is posed by the three-way handshake used to establish a TCP connection. The spoofing attack starts with host $H_i$ sending a first spoofed segment requesting a connection to $H_k$ and pretending to come from $H_j$:

$$s^1 = (a_j, a_k, (p_j, p_k, seq_j^1, 0, \{\mathsf{SYN}\}, x''_{empty})),$$

where $addr^{-1}(a_j) \in H_j$. Host $H_k$, at this point, sends a reply segment:

$$s_2 = (a_k, a_j, (p_k, p_j, seq_k^2, seq_j^1 + 1, \{\mathsf{SYN}, \mathsf{ACK}\}, x''_{empty})).$$

Thus, host $H_i$ must send the segment:

$$s_3 = (a_j, a_k, (p_j, p_k, seq_j^1 + 1, seq_k^2 + 1, \{\mathsf{ACK}\}, x''_{empty}))$$

in order to complete the handshake. Therefore, $H_i$ must know $seq_k^2$. This implies that $H_i$ must be a sniffer of the second datagram, i.e., $H_i \in DS(s^2)$, or that it is able to "guess" the right sequence number. This ability can be achieved analyzing how $H_k$ chooses its sequence numbers when setting up a TCP session, and predicting the value that will be used for the spoofed session. The TCP specification suggests that the number used to generate initial sequence numbers should be increased 250,000 times per second. Some implementations have chosen slower rates. For examples, as described in [21], 4.2 BSD Unix software increases the number used for new connections by 128 each second and by 64 after each new connection setup. Even if modern implementations use faster rates, the attack remains feasible as stated in [4]. A famous break-in [1] was based on this technique.

Once the handshake has been successfully carried out:

1. if the attacker is an "intended" forwarder of every segment between the attacked and the spoofed host, (see (5)) the session may continue seamlessly;
2. if the attacker is a listener of every reply segment between $H_k$ and $H_j$, i.e., called $RS$ the set of all replies from $H_k$ and $H_j$, if:

   $$H_i \in DL(s^i) \text{ for each } s^i \in RS$$

   the session may continue provided that $H_j$ is not able to process $H_k$'s replies;

3. if the attacker cannot access $H_k$'s replies (i.e., it is mounting a "guessing" attack) then, after the initial handshake, the session proceeds as follows:
   - segments from $H_i$ to $H_k$:

     $$s^{ik} = \langle m_1^{ik}, m_2^{ik}, \ldots, m_p^{ik} \rangle \text{ with}$$
     $$pl(m^{ik}) = (a_j, a_k, (p_j, p_k, seq_j^{ik}, seq_k^2 + 1, flags_j^{ik}, x_j''^{ik}))$$

     follow route $r_{ik}$ from $si(m_1^{ik}) \in H_i$ to $di(m_p^{ik}) \in H_k$. Note that the acknowledgment number does not change since $H_i$ does not know the amount of data that have been produced by $H_k$;
   - segments from $H_k$ to $H_j$:

     $$s^{kj} = \langle m_1^{kj}, m_2^{kj}, \ldots, m_q^{kj} \rangle \text{ with}$$
     $$pl(m^{kj}) = (a_k, a_j, (p_k, p_j, seq_k^{kj}, ack_k^{kj}, flags_k^{kj}, x_k''^{kj}))$$

     follow route $r_{kj}$ from $si(m_1^{kj}) \in H_k$ to $di(m_q^{kj}) \in H_j$.

   This communication is just one-way for the attacker, but, in most cases this is enough to breach the security of the attacked host.

# 6 Hijacking

While in spoofing attacks a malevolent host tries to pose as a different (trusted) host for a whole session, in the *hijacking* attack the host tries to interfere with an ongoing session in order to impersonate one partner of the communication with the other and/or vice versa.

## 6.1 Hijacking UDP Interactions

In a UDP hijacking attack a host $H_i$ replies to a legitimate query from $H_j$ to $H_k$, providing false information pretending to come from the host $H_k$.

There are two possibilities:

1. if the attacking host is a forwarder of the hypothetical request $u_h^1$ from $H_j$ to $H_k$, then it may seamlessly impersonate $H_k$, by blocking the request and producing the bogus reply. Thus, if $H_i \in DF(u_h^1)$ the session would be as follows:

   $$u^1 = \langle m_1^1, \ldots, m_p^1 \rangle \text{ with } pl(m_i^1) = (a_j, a_k, (p_j, p_k, x_j''^1)) \text{ and}$$
   $$a_j = addr(si(m_1^1)),\ di(m_p^1) \in H_i;$$
   $$u^2 = \langle m_1^2, \ldots, m_q^2 \rangle \text{ with } pl(m_i^2) = (a_k, a_j, (p_k, p_j, x_k''^2)) \text{ and}$$
   $$si(m_1^2) \in H_i,\ a_j = addr(di(m_q^2));$$

2. if $H_i$ is just a listener of the request $u^1$ from $H_j$ to $H_k$ (which is the same of $u_h^1$ above), then its bogus reply $u_s^2$ must reach the attacked host $H_j$ before the legitimate reply $u^2$ from $H_k$. Unfortunately, in actual TCP/IP networks there are many denial-of-service attacks (e.g., flooding) that are able to slow

down or completely block a host. Therefore, the problem is easily solved. In summary, if $H_i \in DL(u^1)$ the UDP interaction will be:

$$\begin{aligned} u^1 =& \langle m_1^1, \ldots, m_p^1 \rangle \text{ with } pl(m_i^1) = (a_j, a_k, (p_j, p_k, x_j''^1)) \text{ and} \\ & a_j = addr(si(m_1^1)),\ a_k = addr(di(m_p^1)); \\ u_s^2 =& \langle m_1^2, \ldots, m_q^2 \rangle \text{ with } pl(m_i^2) = (a_k, a_j, (p_k, p_j, x_k''^2)) \text{ and} \\ & si(m_1^2) \in H_i,\ a_j = addr(di(m_q^2)). \end{aligned}$$

An example of this attack applied to the Network Information System of Sun Microsystems is given in [14]. The NIS is used in a network of hosts in order to manage and distribute system maps, like host names and user passwords. In particular, when a user logs in a host providing a user name and password, the host's operating system queries the local NIS server in order to get the password file and authenticate the user. An attacker may race with the server and provide a modified password map in a spoofed UDP datagram allowing the user to be authenticated (possibly with extended privileges) with a password specified by the attacker. In another scenario, an attacker may impersonate a NFS server and provide modified version of files in order to breach security.

### 6.2 TCP Session Hijacking

In TCP session hijacking a host $H_i$ tries to interfere with an *existing* TCP session between two hosts $H_j$ and $H_k$. We can, at least, distinguish between two cases: *data injection* and *take over*.

**Injection** In the data injection attack, host $H_i$ sends a single bogus TCP segment to one of the partners of the session (e.g., $H_j$) claiming to come from the other partner (i.e., $H_k$). The segment payload contains a higher-level protocol command that breaches $H_k$ security. The command is interpreted as it was issued by $H_j$.

Let us suppose that the last two segments exchanged between $H_k$ and $H_j$ have been:

$$\begin{aligned} s^t &= (a_j, a_k, (p_j, p_k, seq_j^t, ack_j^t, flags_j^t, x_j''^t)), \\ s^{t+1} &= (a_k, a_j, (p_k, p_j, ack_j^t, seq_j^t + size(x_j''^t), \{\mathsf{ACK}\}, x''_{empty})); \end{aligned}$$

this means that the session is in a "quiet" state and the next segment between $H_j$ and $H_k$ should have the sequence number $seq_j^t + size(x_j''^t)$. Therefore, the attacker $H_i$ produces the following segment:

$$s^{t+2} = (a_j, a_k, (p_j, p_k, seq_j^{t+2}, ack_j^{t+2}, flags_j^{t+2}, x_j''^{t+2})),$$

where: $seq_j^{t+2} = seq_j^t + size(x_j''^t)$, $ack_j^{t+2} = ack_j^t$, and $x_j''^{t+2}$ represents some kind of action aimed at breaking into $H_k$. At this point, $H_k$ replies with an acknowledgment segment:

$$s^{t+3} = (a_k, a_j, (p_k, p_j, ack_j^{t+2}, seq_j^{t+2} + size(x_j''^{t+2}), \{\mathsf{ACK}\}, x''_{empty})).$$

If this segment reaches $H_j$, $H_j$ will produce an acknowledgment message stating that the received segment has a wrong acknowledgment number. The message contains the sequence number $H_j$ believes to be correct, namely, $seq_j^{t+2}$:

$$s^{t+4} = (a_j, a_k, (p_j, p_k, seq_j^{t+2}, ack_j^{t+2}, \{\mathsf{ACK}\}, x''_{empty})).$$

In turn, $H_k$ sends another acknowledgment message containing the acknowledgment number that it considers correct:

$$s^{t+5} = (a_k, a_j, (p_k, p_j, ack_j^{t+2}, seq_j^{t+2} + size(x_j''^{t+2}), \{\mathsf{ACK}\}, x''_{empty})).$$

This message exchange, called *acknowledgment storm*, goes on until a time out expires[5]. Nonetheless, any other attempt to exchange data between $H_i$ and $H_j$ on that session will fail and produce an acknowledgment storm.

In order to mount this attack, $H_i$ must be a sniffer of the at least one of the first two segment in order to determine the correct sequence and acknowledgment numbers to be put in the bogus segment[6]. Therefore it must be: $H_i \in DS(s^t) \cup DS(s^{t+1})$.

**Take over** In a *take over* attack a host $H_i$ gains complete control over an existing session between two hosts $H_j$ and $H_k$. In this attack, extensively described in [16], the attacker creates a *desynchronized* state in the TCP session. In a synchronized TCP session, when all data have been acknowledged, the client (say, $H_j$) sequence number is equal to the server (say, $H_k$) acknowledgment number and vice versa. In a stable desynchronized session the sequence and acknowledgment numbers of the involved parties do not correspond and therefore any data sent by one partner is refused by the other[7]. If the attacker knows the sequence numbers that the parties expect from each other it may filter all traffic by producing the right segments.

In [16], two ways to create a desynchronized state are described. In the first case, the attacker resets the current session between $H_j$ and $H_k$ and immediately opens a new connection with $H_k$. In this scenario, the three-way TCP setup handshake (see Section 2.3) and subsequent data exchange have determined the current sequence numbers $seq_j$ and $seq_k$. Then, the attacker $H_i$ sends a reset segment to $H_k$, immediately followed by a new connection request. Both segments appear to come from $H_j$:

$$s^t = (a_j, a_k, (p_j, p_k, seq_j, seq_k, \{\mathsf{RST}\}, x''_{empty}));$$
$$s^{t+1} = (a_j, a_k, (p_j, p_k, seq_j^{t+1}, 0, \{\mathsf{SYN}\}, x''_{empty}));$$

[5] Following the actual TCP specification, the process should go on indefinitely, but since acknowledgment messages with empty payloads are not retransmitted in case of errors, the storm stops when the traffic that has been produced leads to dropping some of the packets.

[6] From a theoretical point of view, this attack could be composed with the one described by Morris in [21] in order to be able to mount an attack from a host that it is not a listener of traffic between the attacked hosts. However, it would be rather difficult to guess the correct sequence number in an established session over which random traffic may occur.

[7] Actually the sequence number must be out of a *window* of acceptable values [26], but we will assume that the session is sufficiently desynchronized.

$H_k$ then replies to $s^{t+1}$ with an acknowledgment/synchronization segment (second step of the setup):

$$s^{t+2} = (a_k, a_j, (p_k, p_j, seq_k^{t+2}, seq_j^{t+1} + 1, \{\mathsf{SYN}, \mathsf{ACK}\}, x''_{empty})).$$

The attacker replies to this segment with the last step of the setup:

$$s^{t+3} = (a_j, a_k, (p_j, p_k, seq_j^{t+1} + 1, seq_k^{t+2} + 1, \{\mathsf{ACK}\}, x''_{empty})).$$

If segment $s^{t+2}$ reaches $H_j$, it will start an acknowledgment storm between $H_j$ and $H_k$:

$$s^{t+4} = (a_j, a_k, (p_j, p_k, seq_j, seq_k, \{\mathsf{ACK}\}, x''_{empty}));$$
$$s^{t+5} = (a_k, a_j, (p_k, p_j, seq_k^{t+2} + 1, seq_j^{t+1} + 1, \{\mathsf{ACK}\}, x''_{empty}));$$
$$\vdots$$

At this point the session is desynchronized. The attacker knows all the "right" sequence and acknowledgment numbers, and, therefore, he/she may filter/change all traffic. In fact, when $H_j$ sends a segment:

$$s^l = (a_j, a_k, (p_j, p_k, seq_j, seq_k, flags_j^l, x_j''^l)),$$

the segment is refused by $H_k$ and generates an acknowledgment storm. But the attacker sends a modified version of $s^l$:

$$s^{l+1} = (a_j, a_k, (p_j, p_k, seq_j^{t+1} + 1, seq_k^{t+2} + 1, \{\mathsf{ACK}\}, x_j''^l)),$$

which $H_k$ happily accepts because it contains the right acknowledgment and sequence numbers. The acknowledgment of the segment is processed accordingly. Note that the users at the endpoints of the virtual circuit do not perceive any abnormal behavior. However, the attacker has complete control over the connection and can modify or insert data in the stream.

The second way to create a desynchronized connection consists in sending to the communication partners some data that increase the sequence numbers associated to the virtual circuit. Thus, once $H_j$ and $H_k$ have setup a session and exchanged some data so that their sequence numbers are $seq_j$ and $seq_k$ respectively, the attacker $H_i$ sends to $H_k$:

$$s^t = (a_j, a_k, (p_j, p_k, seq_j, seq_k, flags_j^t, x_j''^t)).$$

$H_k$ will send an acknowledgment segment:

$$s^{t+1} = (a_k, a_j, (p_k, p_j, seq_k, seq_j + size(x_j''^t), flags_k^{t+1}, x_k''^{t+1})),$$

and start an acknowledgment storm. In the meantime, $H_i$ sends some data to $H_j$ pretending to come from $H_k$:

$$s^{t+2} = (a_k, a_j, (p_k, p_j, seq_k, seq_j, flags_k^{t+2}, x_k''^{t+2})).$$

$H_j$ will acknowledge the segment and start another storm. At this point, the session is desynchronized and the attacker may filter or inject any traffic. In

order to make this attack less detectable, the data used to desynchronize the session, namely, $x_j''^t$ and $x_k''^{t+2}$, should have no effect on the upper-layer protocol (e.g., some kind of no-operation command).

In order to accomplish take over with both desynchronization methods, the attacker must be a sniffer of all the segments of the session between $H_j$ and $H_k$, i.e., called $ES$ the set of such segments, we must have: $H_i \in DS(s^i)$ for each $s^i \in ES$.

## 7 Using The Model

The proposed network model can be used to support network security analysis in a number of different ways.

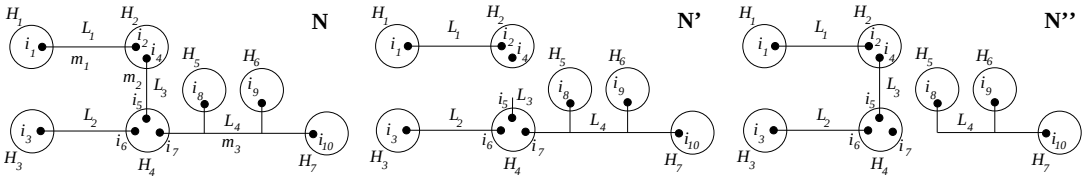
- *Better understanding of the vulnerabilities*: modeling attacks it is possible to define formally the prerequisites to mount each type of attack and the messages produced during the attack.
- *Support for detection of attacks*: given a particular vulnerability, it is possible to devise which are the conditions sufficient to detect particular attack scenarios, and which are the patterns to be looked for.
- *Support for protection from attacks*: given a particular vulnerability it is possible to find which are the configurations that provide protection from attacks of the given type and if such configurations exist.
- *Support for testing and certification of networks*: given a particular network topology, the model can be used in order to automatically produce testing procedures that verify the existing protections. In addition, the model can be used by solution vendors to validate and certify security protections.

### 7.1 Detection

Defining exactly which messages are produced during attacks and where they appear provides hints for network intrusion detection systems [10, 2, 3]. The model allows the security designer to determine which are the message patterns that are to be looked for in order to flag out occurring intrusion attempts and where, in the network topology, probes have to be placed in order to detect the largest range of attacks. The model proposed here has been extensively used in designing and developing the STAT system [30, 31]. In the following, we describe how the model can be used to determine the conditions that are necessary to detect the attacks described before.

**Spoofing** As stated in Section 5.1, usually the receiver of a spoofed IP datagram is not able to detect the attack. In general, IP spoofing is detectable only in particular topologies. More precisely, given a single message belonging to a spoofed IP datagram, the corresponding datagram may be considered spoofed if, in the network obtained removing the message source interface from the corresponding link, there is no path between the interface corresponding to the datagram source address and the message source interface. Thus, given a network $N = (H_1, \ldots, H_p, L_1, \ldots, L_q)$ and a message $m_i \in d$, $d$ can be considered spoofed if in:

$$N' = N - \{al(m_i)\} \cup \{al(m_i) - \{si(m_i)\}\}$$




$$d = \langle m_1, m_2, m_3 \rangle$$
$$m1 = (i_1, i_2, (a_3, a_{10}, p))$$
$$m2 = (i_4, i_5, (a_3, a_{10}, p))$$
$$m3 = (i_7, i_{10}, (a_3, a_{10}, p))$$
with $a_j = addr(i_j), j = 1, 2, \ldots, 10$.

**Fig. 3.** IP spoofing example.

there is no path between $addr^{-1}(dsa(m_i))$ and $si(m_i)$.

For example, consider Figure 3. Host $H_1$ tries to impersonate host $H_3$ toward host $H_7$ with the spoofed datagram $d = \langle m_1, m_2, m_3 \rangle$. Host $H_4$, that sees message $m_2$, is able to detect the occurring spoofing attack since in network $N'$ there is no path between $i_3 = addr^{-1}(dsa(m_2))$ and $i_4 = si(m_2)$. On the contrary, host $H_5$ is not able to detect spoofing since it accesses only message $m_3$ and in network $N''$ there is a path between $i_3$ and $i_7$.

We define a *reachability* function *reach* that, given a network $N = \{E_1, \ldots, E_m\}$ and an interface $i_j \in E_l$ for some $l$, returns the set of interfaces that are reachable from $i_j$:

$$reach : I \times \wp(\wp(I)) \rightarrow \wp(I), \text{ defined by}$$
$$reach(i_j, \{E_1, E_2, \ldots, E_m\}) = \{i_k \mid \exists r = \langle E_{r_1}, E_{r_2}, \ldots, E_{r_n} \rangle \in R$$
$$\text{such that } i_j \in E_{r_1} \wedge i_k \in E_{r_n}\}$$

Given a host $H_i$, we can build its *spoofing detection matrix* $SDM(H_i)$ that has rows labeled with the links the host is connected to and columns labeled with all network addresses. The matrix contains, for each row, a 1 for every address whose spoofing the host can detect from messages appearing on that link and a 0 otherwise. Each row is built in the following way: given the associated link $L_k$, for each interface $i_j \in L_k \wedge i_j \notin H_i$ consider the network $N_{jk}$ obtained removing $i_j$ from $L_k$ and associate a 0 to the columns corresponding to the elements of $Im_{addr_{|I_{jk}}}$ where $I_{jk} = reach(i_j, N_{jk})$.

For example, let us consider the network $N$ in Figure 3. $SDM(H_4)$ is:

| | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ | $a_8$ | $a_9$ | $a_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $L_2$ | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $L_3$ | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| $L_4$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

From the matrix above we can induce that host $H_4$ can detect every attempt coming from link $L_4$ to spoof any address.

A UDP or TCP spoofing attack may also be detected by means of *orphan datagrams*. Such datagrams appear when a host $H_i$ is impersonating a host $H_j$ toward a third host $H_k$ (actually $H_i$ is using its interface $i_i$ to impersonate one of the addresses of $H_j$, say $a_j$, when communicating with one of the addresses of $H_k$, say $a_k$) and it is not able to block the reply traffic from $H_k$ to $H_j$. Thus, there will be particular links or hosts on the network on which the interaction/session datagrams from $H_k$ will appear and the ones from $H_j$ will not[8]. More precisely, given the set $R_{kj}$ of all routes from $i_k = addr^{-1}(a_k)$ to $i_j = addr^{-1}(a_j)$:

$$R_{kj} = \{\langle E_1, \ldots, E_n\rangle \in R \mid i_k \in E_1 \wedge i_j \in E_n\}, \tag{6}$$

we extract the subset $E_{kj}$ of edges that are common to all routes:

$$E_{kj} = \bigcap_{r \in R_{kj}} r, \tag{7}$$

and, given the set $C_{ik}$ of all edges that appear at least in one route from $i_i$ to $i_k$:

$$C_{ik} = \bigcup_{r \in R_{ik}} r, \text{where } R_{ik} \text{ is defined by (6)},$$

we obtain the set of edges $O_{ijk}$:

$$O_{ijk} = E_{kj} - C_{ik}.$$

Orphan datagrams can be detected by hosts that are in $O_{ijk}$ or that are connected to a link in $O_{ijk}$:

$$H^o_{ijk} = \{H_o \in H \mid H_o \in O_{ijk} \vee \exists L_o \in O_{ijk} \text{ such that } H_o \in C(L_o)\}.$$

For example, let us consider Figure 4. Host $H_3$ is impersonating host $H_8$ toward host $H_1$. Spoofed traffic from $H_3$ to $H_1$ follows route $\langle L_2, H_2, L_1\rangle$ while traffic from $H_1$ to $H_8$ follows route $\langle L_1, H_2, L_3, H_6, L_5, H_7, L_6\rangle$. Orphan datagrams can be detected from hosts that belong to route $\langle L_5, H_7, L_6\rangle$ or that are connected to links appearing in the route. Even if host $H_4$ is able to see replies from $H_1$ to $H_8$, it is not able to distinguish between orphan datagrams and replies to legitimate traffic that has followed a different route (e.g., $\langle L_6, H_7, L_5, H_6, L_4, H_5$,-$L_2, H_2, L_1\rangle$). Different, $H_7$ is able to identify datagrams as orphans since it can be sure that no traffic from $H_8$ to $H_1$ was generated before the orphan datagrams.

**Hijacking** In the case of UDP hijacking we have the dual situation with respect to spoofing: instead of "orphan" datagrams we have "twin" datagrams. These datagrams are the false reply (say, produced by the attacker $H_i$ from its interface

[8] Actually, in case of UDP spoofing, since there is no way to determine the membership of a single datagram to an interaction. The datagram must contain some kind of protocol-specific information that identifies the datagram as a reply in an interaction. For example, examining UDP datagrams of an NFS operation we can distinguish between request and reply messages.

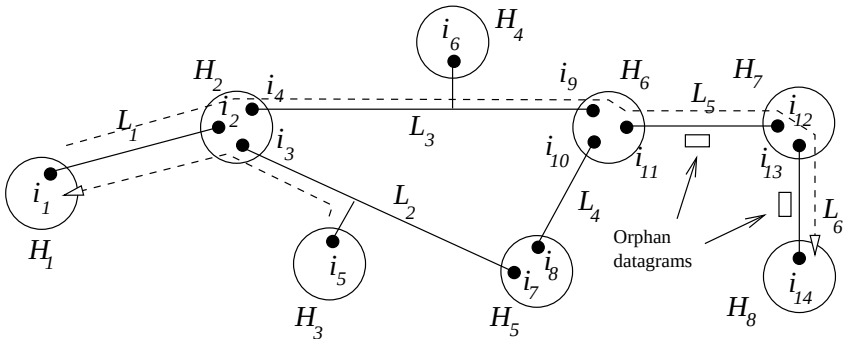



**Fig. 4.** Orphan datagrams.

$i_i$) followed by the legitimate response (say, produced by the server $H_k$ from the address $a_k$) that follow a request (say, produced by the attacked host $H_j$ from its address $a_j$). Given the set $E_{ij}$ of all edges that appear in all routes from $i_i$ to $i_j = addr^{-1}(a_j)$ (defined by (7)) and the set $E_{kj}$ of all edges routes from $i_k = addr^{-1}(a_k)$ to $i_j$, we consider the set: $T_{ijk} = E_{ij} \cap E_{kj}$.

Twin datagrams can be detected by hosts in $T_{ijk}$ or connected to links in $T_{ijk}$, i.e.,:

$$H^t_{ijk} = \{H_t \in H \mid H_t \in T_{ijk} \vee \exists L_t \in T_{ijk} \text{ such that } H_t \in C(L_t)\}.$$

For example, suppose host $H_{10}$ is requesting some kind of service to host $H_1$ (see Figure 5 (a)). Host $H_3$ hijacks the UDP interaction and produces a forged reply directed to $H_{10}$ and pretending to come from $H_1$. In addition, $H_1$ sends the legitimate reply to $H_{10}$. As can be seen in Figure 5 (b), twin datagrams appear on links $L_5$ and $L_8$ and will be forwarded by hosts $H_6$, $H_7$, and $H_9$. Host $H_8$ does not see the twin datagrams since it is not connected to a link that is a forced passage from both $H_1$ and $H_3$ to $H_{10}$. In principle, if both twin datagrams would have followed the route than includes $L_6$, $H_8$ could have detected the hijacking.

Detecting TCP session hijacking attacks is easy because such attacks produce the *acknowledgment storms* that are easily detectable by every hosts that is a listener or forwarder of all the traffic occurring between the attacked hosts $H_j$ and $H_k$. Therefore, assuming that the hijacked session is occurring between addresses $a_j = addr(i_j)$ of $H_j$ and $a_k = addr(i_k)$ of $H_k$, the set of hosts that can detect the acknowledgment storm is:

$$J_{jk} = \{H_l \in H \mid H_l \in E_{jk} \vee \exists C_l \in E_{jk} \text{ such that } H_l \in C(L_l)\},$$

where $E_{jk}$ is defined by (7).

### 7.2 Protection

The proposed formal model, defining the prerequisites to mount attacks and the conditions to be verified in order to be able to detect attacks, provides support to the design of security protections.

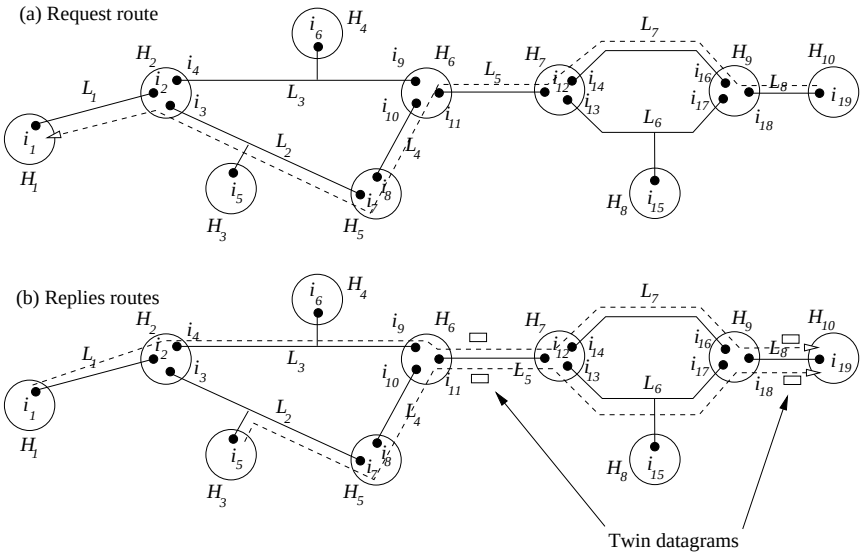



**Fig. 5.** Twin datagrams.

**Sniffing** The only effective means to protect from sniffing is the encryption of traffic. Such protection must be implemented at the application level (e.g., [33]), since the TCP/IP protocol suite does not provide any mechanisms to protect the transmitted data.

Nonetheless, the model may provide useful information. When sending a datagram to $H_k$, $H_j$ (actually its network software) may use the network topology description in order to compute all possible routes to destination and the corresponding possible sniffers, following the procedures described in Section 3. Then, $H_j$ may decide to choose a specific route (e.g., by using source routing) adopting some heuristics or quality-of-service parameter. For example, $H_j$ may define a *listener vector* $SV(H_j)$ that for each host $H_i \neq H_j$ associates a value representing a certain level of trust. Then, when comparing the possible routes $H_j$ may choose to maximize the average trust of the path or not to send the packet if hosts with a too low level of trust are listeners of the packet. The choice could also be supported by a security model of the transmitted data.

**Spoofing** The model can be used as a basis to build effective protections from spoofing attacks. For example, given a particular protocol (e.g., NFS) one can build the corresponding *trust vector TM* for a given host. The vector identifies the set of addresses that the host considers *trusted*, i.e., the set of addresses from which requests are accepted without further authentication. The trust vector can be used in conjunction with the *SDM* matrix of each host in order to determine which hosts can successfully build a spoofing attack for that particular protocol without being detected. For example, let us suppose that host $H_7$ in Figure 3 is an NFS server exporting its file systems to $H_5$, $H_6$ and $H_1$ (actually, to the

addresses associated to their interfaces, namely $a_8 = addr(i_8)$, $a_9 = addr(i_9)$, and $a_1 = addr(i_1)$). Thus $H_7$'s trust vector for the NFS protocol is:

| $a_1$ | $a_2$ | $a_3$ | $i_4$ | $i_5$ | $i_6$ | $i_7$ | $i_8$ | $i_9$ | $i_{10}$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

Given this scenario, we can analyze the spoofing detection matrix of all the hosts involved and find which is the most suitable host to detect spoofing attacks. In our case, host $H_4$ is able to detect every possible attempt to spoof addresses $a_8$ and $a_9$, but it is not able to detect the attempts to spoof address $a_1$ appearing on link $L_3$. Thus, in order to cover all possible attacks, we have to use both hosts $H_2$ and $H_4$ in order to protect the server.

**Hijacking** As in the case of sniffing, protection from hijacking attacks must be implemented at the application level. If the traffic is encrypted, the attacker can filter every byte exchanged between the partners of the communication but it is not able to insert meaningful commands aimed at breaking the security of the endpoints of the connection. Nonetheless, an attacker may use injection in order to realize a "denial of service" attack, preventing any information exchange.

### 7.3 Testing and validation

Most testing of network security tools [23, 15] use *ad hoc* techniques that are based neither on a formal model nor on information as network topology and trust.

The proposed model can be used as a basis for the automatic generation of test cases for network security software (e.g., firewalls). Given a particular network topology and the types of protection implemented, the model can be used in order to determine a set of test attacks and from where such attacks must be carried out. In addition, the model can be used by solution vendors in order to certify the offered protections. The main problem for network security applications is that their certification is often done once for all and is not performed in the context of the particular network to protect. The model can support a sound, formal certification that is related to the particular topology of the protected network. This way, solution vendors could show that given the particular topology and a set of attacks described on the model, the network is immune to such attacks.

## 8 Conclusions and Future Work

Security is a growing concern. The Internet, the world-wide TCP/IP network, is becoming an important part of everyday life, and people rely on services offered by the TCP/IP protocol suite in order to carry out personal communication and commercial transactions. Despite its critical infrastructure role, the TCP/IP suite is insecure. There are many *ad hoc* techniques used to protect TCP/IP networks from known attacks, but still there is no underlying formal framework.

We have presented a formal model of networks and of TCP/IP networks in particular. We have used the model to describe some well-known vulnerabilities of the TCP/IP protocol suite, namely *sniffing*, *spoofing*, and *hijacking*. Then,

we have shown how the model of the network and of the attacks can be used as a basis to perform network security analysis. In particular we showed how the topology constraints derived from the modeling of the attacks can be used to determine the placement of intrusion detection system and to identify flawed configurations.

Future work will be aimed to model more attack techniques, in particular those exploiting the vulnerability of routing protocols, where the topological characterization of the network is an integral part of the attack analysis. We are also working on extending the model in order to be able to describe some state associated to hosts (e.g., routing tables) and to take into account the dynamic aspects of networks (e.g., dynamically assigned addresses). We also plan to use the model as a basis to build toolsets for topology-aware testing of firewall systems.

## References


1. T. Shimomura amd J. Markoff. *Takedown.* Hyperion, 1996.
2. S. Axelsson. Intrusion Detection Systems: A Taxomomy and Survey. Technical Report 99-15, Dept. of Computer Engineering, Chalmers University of Technology, Sweden, March 2000.
3. R. Bace and P. Mell. Special Publication on Intrusion Detection Systems. Technical Report SP 800-31, National Institue of Standards and Technology, November 2001.
4. S. Bellovin. Security Problems in the TCP/IP Protocol Suite. *Computer Communications Review*, 19(2), 1990.
5. C. Berge. *Hypergraphs.* North-Holland, 1989.
6. CERT. Cert advisories. http://www.cert.org, 2003.
7. B. Chapman and E. Zwicky. *Building Internet Firewalls.* O'Reilly & Associates, 1995.
8. W. Cheswick and S. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker.* Addison-Wesley, 1994.
9. Computer Emergency Response Team. IP Spoofing Attacks and Hijacked Terminal Connections. CA-95:01, January 1995.
10. H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, 1999.
11. R. Fielding et al. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999.
12. International Organization for Standardization. Information Processing Systems - Open Systems Interconnection. International Standard, 1986.
13. A Freier, P. Karlton, and P. Kocher. The ssl protocol version 3.0. Internet draft `draft-freier-ssl-version3-02.txt`, November 1996.
14. D.K. Hess, D.R. Safford, and U.W. Pooch. A Unix Network Protocol Security Study: Network Information Service. Technical report, Texas A&M University, November 1992.
15. ISS. Realsecure 6.5. http://www.iss.net/, February 2002.
16. L. Joncheray. A Simple Active Attack Against TCP. Technical report, Merit Network Inc., April 1995.
17. J. Kohl and C. Neuman. The Kerberos Authentication Service (V5). RFC 1510, September 1993.
18. G. Malkin. Rip version 2. IETF RFC 2453, Nov 1998.
19. Sun Microsystems. NFS: Network File System Protocol Specification. RFC 1094, 1989.
20. P. Mockapetris. Domain Name System. RFC 1034, November 1987.
21. R.T. Morris. A Weakness in the 4.2BSD UNIX TCP/IP Software. Technical report, AT&T Bell Laboratories, February 1985.

22. Mudge. Antisniff 1.1. http://www.L0pht.com/antisniff/, May 2000.
23. Nessus homepage. http://www.nessus.org/, 2003.
24. J. Postel. User Datagram Protocol. RFC 768, August 1980.
25. J. Postel. Internet Protocol. RFC 792, 1981.
26. J. Postel. Transmission Control Protocol. RFC 793, September 1981.
27. J. Postel. Simple Mail Transfer Protocol. RFC 821, 1982.
28. Y. Rekhter and T. Li. A border gateway protocol 4 (bgp-4). IETF RFC 1654, Mar 1995.
29. D. Song. Dsniff version 2.3. http://naughty.monkey.org/ dugsong/dsniff/, Feb 2003.
30. G. Vigna and R.A. Kemmerer. NetSTAT: A Network-based Intrusion Detection System. *Journal of Computer Security*, 7(1):37–71, 1999.
31. G. Vigna, R.A. Kemmerer, and P. Blix. Designing a Web of Highly-Configurable Intrusion Detection Sensors. In W. Lee, L. Mè, and A. Wespi, editors, *Proceedings of the $4^{th}$ International Symposiun on Recent Advances in Intrusion Detection (RAID 2001)*, volume 2212 of *LNCS*, pages 69–84, Davis, CA, October 2001. Springer-Verlag.
32. R. Zakon. Hobbes' internet timeline. http://www.zakon.org/robert/internet/timeline/, February 2003. Version 6.0.
33. Philip Zimmerman. *PGP User's Guide*, March 1993.