Chapter 6

Network Security II

Contents

6.1 The Application Layer and DNS

The physical, link, network, and transportation layers provide a basic underlying network infrastructure that allows applications to communicate with each other. It is in the application layer that most of the action of the Internet takes place.

6.1.1 A Sample of Application-Layer Protocols

There are many application-layer protocols designed to perform a number of important tasks at Internet-scale, including the following:

- ***Domain name system*** (***DNS***). This is the protocol that allows us to use intuitive domain names to refer to Internet hosts rather than using IP addresses. Most application programs and other application-layer services rely on DNS.
- ***Hypertext transfer protocol*** (***HTTP***). This is the protocol used to browse the Web and is discussed in detail in Section 7.1.1.
- ***SSL/TLS***. This is the protocol used for secure, encrypted browsing (i.e., with HTTPS) and is also discussed in Section 7.1.2.
- ***IMAP/POP/SMTP***. These are protocols that make Internet email possible. They are discussed in Section 10.2.
- ***File transfer protocol*** (***FTP***). This is an old, but still used, protocol that provides a simple interface for uploading and downloading files. It does not encrypt data during transfer.
- ***SOAP***. This is a more recent protocol for exchanging structured data as a part of the web services paradigm.
- ***Telnet***. This is an early remote access protocol. Like FTP, it doesn't encrypt connections.
- ***SSH***. This is a more recent secure remote access and administration protocol, and is discussed in Section 6.3.1.

Each application-layer protocol comes with its own security considerations, and an entire book could be written on the vast number of application-layer protocols. In this section, we focus on one of the most commonly used protocols, DNS, since it is one of the pillars of the architecture of the Internet itself.

6.1.2 The Domain Name System (DNS)

The ***domain name system***, or ***DNS***, is a fundamental application layer protocol that is essential to the functioning of the Internet as we know it today. DNS is a protocol that sits "behind the scenes" for every web browser and is responsible for resolving ***domain names***, such as www.example.com, to IP addresses, such as 208.77.188.166. (See Figure 6.1.)

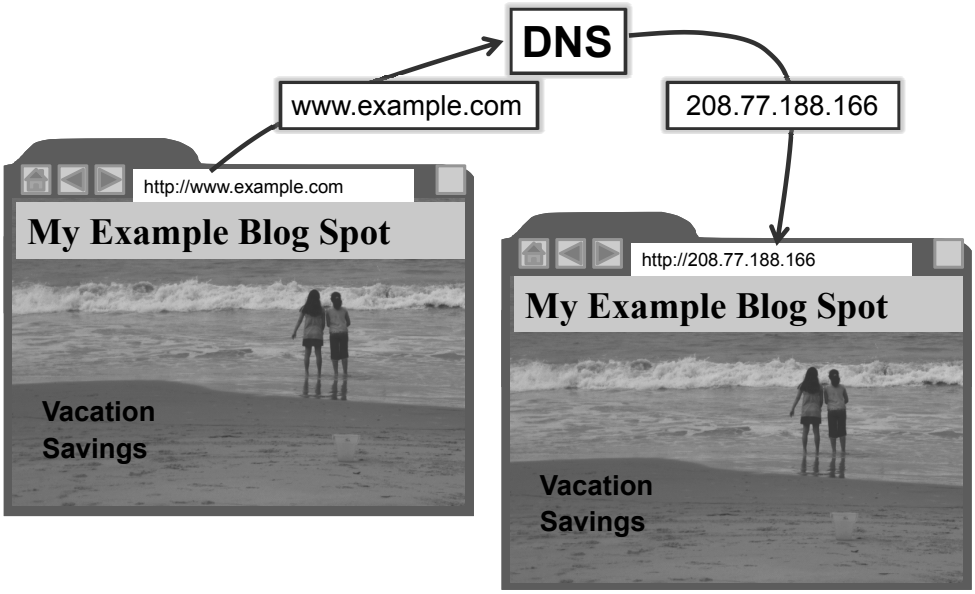

Figure 6.1: The DNS protocol performs a lookup for domain name www.example.com to find the IP address associated with this domain. (Image by Karen Goodrich; used with permission.)

It is hard to imagine surfing the net without DNS, in fact. For instance, would the Internet still be popular if we had to tell our friends about the video we just watched on 74.125.127.100?

Domain names are arranged in a hierarchy that can be read by examining a domain name from right to left. For example, www.example.com has a ***top-level domain*** (***TLD***) of com, with example.com being a ***subdomain*** of com, and www.example.com being a subdomain of example.com. More formally, domain names form a rooted tree, where each node corresponds to a domain and the children of a node correspond to its subdomains. The root is the empty domain name and the children of the root are associated with top-level domains.

Domain Name Registration

There are two primary types of top-level domains in use today:

- ***Generic top-level domains***, such as the popular domains .com, .net, .edu, and .org
- ***Country-code top-level domains***, such as .au (Australia), .de (Germany), .it (Italy), and .pt (Portugal), with use restricted to entities within a specific country

Domain names are registered and assigned by ***domain-name registrars***, which are organizations accredited by the ***Internet Corporation for Assigned Names and Numbers*** (***ICANN***), the same group responsible for allocating IP address space, or a country-code top-level domain that has been granted authority to designate registrars. Web site owners wishing to register a domain name can contact a domain-name registrar to reserve the name on their behalf.

The registration process itself is pretty simple. Other than a small fee charged by a domain-name registrar, the rest of the registration process simply involves providing some contact information. This information is often publicly available, however, and can be a source of valuable information for an attacker.

For example, common system utilities such as whois can be used to retrieve the contact information of the owner of a particular domain, which might then be used to initiate a social engineering attack. To avoid disclosing personal details via this information, some web site owners choose to use anonymous domain registration services that specifically do not publish contact information for their customers. Unfortunately, this use of anonymity can sometimes be abused.

Because of the revenue potential of memorable domain names, a practice known as ***cybersquatting*** or ***domain squatting*** has become commonplace. In such a scenario, a person registers a domain name in anticipation of that domain being desirable or important to another organization, with the intent of selling the domain to that organization for what can sometimes be a significant profit. Some cybersquatters go so far as to post negative remarks or accusations about the target organization on this page to further encourage the target to purchase the domain in defense of its reputation. Such practices are now illegal under U.S. law, but it is often difficult to determine the line between malicious intent and coincidental luck in choosing marketable domain names.

How DNS is Organized

The hierarchical nature of domain names is reflected in the way the Internet infrastructure supporting the DNS system works. That is, to ***resolve*** a domain name to its corresponding IP address, the DNS hierarchy is used to query a distributed system of DNS servers, known as ***name servers***. At the top of the name-server hierarchy are the ***root name servers***, which are responsible for top-level domains, such as .com, .it, .net, and .org. Specifically, the root name servers store the ***root zone database*** of records indicating the ***authoritative name server*** of each top-level domain. This important database is maintained by ICANN. The name servers of each top-level domain are managed by government and commercial organizations. For example, the name servers for the .com TLD are managed by VeriSign, a company incorporated in the U.S., while the name servers for the .it TLD are managed by the Italian National Research Council, an Italian government organization. In turn, the TLD name servers store records for the authoritative name servers of their respective subdomains. Thus, the authoritative name servers are also organized in a hierarchy. (See Figure 6.2.)

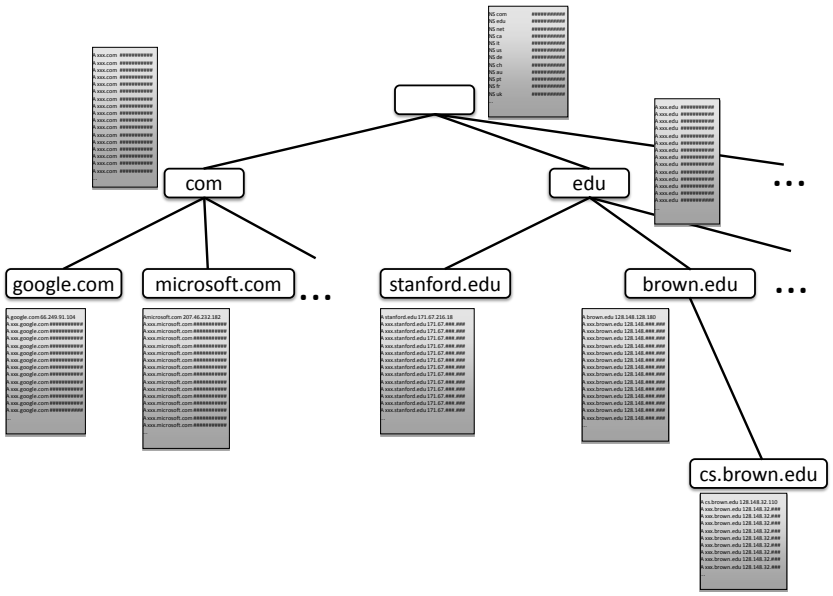

Figure 6.2: The hierarchical organization of authoritative name servers. Each name server stores a collection of records, each providing the address of a domain or a reference to an authoritative name server for that domain.

How DNS Queries Work

When a client machine wishes to resolve a domain name such as www.example.com to an IP address, it contacts a designated name server assigned to the machine. This designated name server can be, for example, a name server of the corporate network to which the client machine belongs, or a name server of the Internet service provider. The designated name server handles the resolution of the domain name and returns the result to the client machine, as follows.

First, the designated name server issues a DNS query to a root name server. The root server then responds with the address of the server that is authoritative for the next level of the hierarchy—in this example, it would reply with the address of the name server responsible for the .com top-level domain name. On querying this next-level server, it would respond with the address of the name server responsible for the next subdomain, which in this case is example.com. This sequence of requests and responses continues until a name server responds with the IP address of the requested domain. This final name server is therefore the authoritative responder for the requested domain name, which in this case is www.example.com.

The process of domain-name resolution is depicted in Figure 6.3.

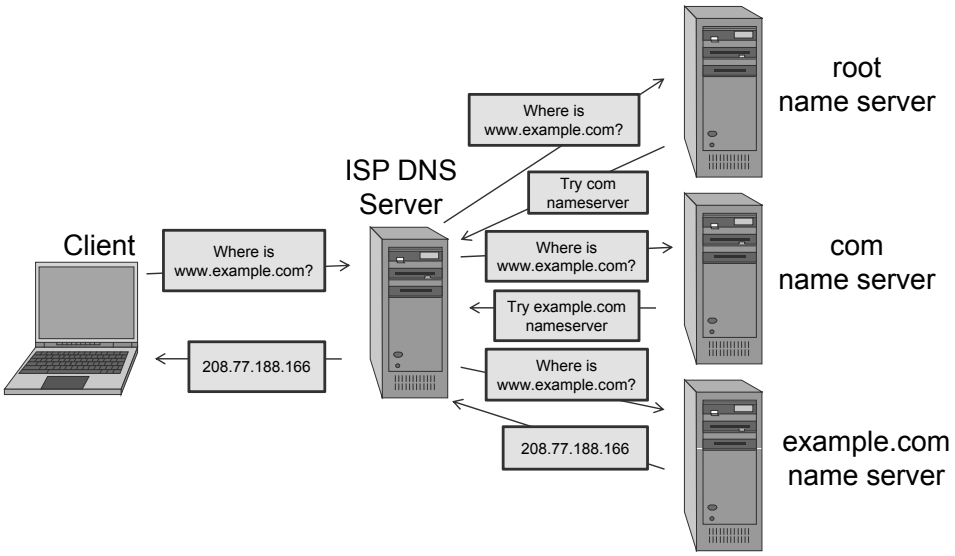

Figure 6.3: A typical execution of a DNS query. The client machine queries a designated name server, such as a name server of its service provider. The designated name server in turn queries a root name server, then a top-level domain name server, and finally the authoritative name server for the requested domain. Once the intermediate name server resolves the domain name, it forwards the answer to the client machine.

DNS Packet Structure

DNS queries and replies are transmitted via a single UDP packet, with TCP being used as a substitute for requests or replies exceeding 512 bytes. The standard UDP packet used for DNS consists of a header, a query part, and an answer part.

The header is formated as follows:

- The header includes a 16-bit ***query identifier***, also called ***transaction identifier***, which identifies the query and response.

The query part, in turn, consists of the following:

- The query part is a sequence of "questions" (usually just one), each consisting of the domain name queried and the type of record requested. The query ID is selected by the client sending the query and is replicated in the response from the server.

The answer part consists of a sequence of DNS records, each of consisting of the following fields:

- The NAME field is of variable length and contains a full domain name.
- The 2-byte TYPE field indicates the type of DNS record. A standard domain-to-address resolution is described by an A record, but other types exist as well, including NS records (providing information about name servers), MX records (providing information about email resolution), and several other less commonly used record types.
- The 2-byte CLASS field denotes the broad category that the record applies to, such as IN for Internet domains.
- The 4-byte TTL field specifies how long a record will remain valid, in seconds.
- The 2-byte RDLENGTH field indicates the length of the data segment, in bytes.
- The variable-length RDATA segment includes the actual record data. For example, the RDATA segment of an A record is a 32-bit IP address.

DNS Caching

Since DNS is a central service utilized by billions of machines connected to the Internet, without any additional mechanisms, DNS would place an incredible burden on high-level name servers, especially the root name servers. In order to reduce DNS traffic and resolve domain names more efficiently, DNS features a caching mechanism that allows both clients and lower-level DNS servers to keep a ***DNS cache***, a table of recently receivd DNS records. A name server can then use this cache to resolve queries for domain names it has recently answered, rather than consuming the resources of higher-level name servers. This caching system therefore overcomes the problem of massive amounts of traffic directed at root name servers by allowing lower-level name servers to resolve queries.

Caching changes how DNS resolution works. Instead of directly querying each time a root name server, the designated name server first checks its cache and returns to the client the requested IP if a record is found. If not, the designated name server queries the root name server and resolves the domain name as described above, caching the result as it is returned to the client. A value known as the ***time-to-live*** (***TTL***) determines how long a DNS response record remains in a DNS cache. This value is specified in the DNS response, but administrators can configure local settings that override the provided TTL values. Once a cached record has expired, the query process resorts back to asking a higher-level name server for a response.

Some operating systems maintain a local DNS cache on the machine. If a valid record is found for the desired domain, then this record is used and no DNS queries are issued. The details of DNS caching depend on the chosen operating system and application. For example, Windows features its own DNS cache, while many Linux distributions do not. They opt to query predetermined name servers for each resolution instead. The DNS cache on a Windows system can be viewed by issuing the command `ipconfig /displaydns` at a command prompt. In general, web browsers are responsible for extracting a user-supplied domain name and passing it to the operating system's networking component, which handles the sending of a corresponding DNS request. The reply will then be received by the operating system and passed back to the browser. At this stage, if the operating system has its own DNS cache, it stores the DNS reply information in the cache before passing it back to the application. DNS caches maintained by operating systems have privacy implications for users. Namely, even if the user deletes the browsing history and cookies, the DNS cache will preserve evidence of recently visited sites, which could be unveiled by forensic investigation.

In addition, several cross-platform browsers, including Firefox, support their own DNS caches. However, Internet Explorer, which is intended to run on Windows, does not implement this feature because Windows has its own cache.

Another challenge in the resolution of domain names is the possibility of infinite loops. Suppose in the example above, the .com name server replied indicating that the authoritative name server for the example.com domain is ns1.example.com. DNS responses that delegate to other name servers identify these name servers by name, rather than by IP address, so an additional DNS request is required to resolve the IP address of ns1.example.com. However, because the name server is both a subdomain of example.com and its authoritative name server, there is a circular dependency that cannot be resolved. In order to resolve example.com, ns1.example.com must be resolved first, but in order to resolve ns1.example.com, example.com must first be resolved. To break these loops, responses include glue records that provide enough information to prevent these dependencies. In this example, the .com name server would include a glue record resolving ns1.example.com to its IP address, giving the client enough information to continue.

One can experiment with DNS resolution with the help of several command-line tools. On Windows, nslookup can be used at a command prompt to issue DNS requests. On Linux, users may use either nslookup or dig, as depicted in Figure 6.4.

```
cslab % dig @4.2.2.2 www.example.com
; <<>> DiG 9.6-ESV-R1 <<>> @4.2.2.2 www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29228
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

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

;; ANSWER SECTION:
www.example.com.        43200   IN      A       192.0.32.10

;; Query time: 88 msec
;; SERVER: 4.2.2.2#53(4.2.2.2)
;; WHEN: Thu Jul 15 01:17:47 2010
;; MSG SIZE  rcvd: 49
```

Figure 6.4: Using the dig tool to issue a DNS query for domain www.example.com to the root name server at IP address 4.2.2.2.

6.1.3 DNS Attacks

By relying on DNS to resolve domain names to IP addresses, we place a large degree of trust in the fact that DNS requests are resolved correctly. When we navigate to www.example.com, for instance, we expect to be directed to the IP address actually associated with that domain name.

Pharming and Phishing

Consider, however, what could happen if DNS were somehow subverted so that an attacker could control how DNS requests resolve. Because DNS is so central to how domain names are used to navigate the Web, such a subversion would cause the safety of the Web to be compromised. An attacker could cause requests for web sites to resolve to false IP addresses of his own malicious servers, leading the victim to view or download undesired content, such as malware. Such an attack is known as ***pharming***.

One of the main uses of pharming is to resolve a domain name to a web site that appears identical to the requested site, but is instead designed for a malicious intent. Such an attack is known as ***phishing*** and it can be used to try to grab usernames and passwords, credit card numbers, and other personal information. (See Figure 6.5.)

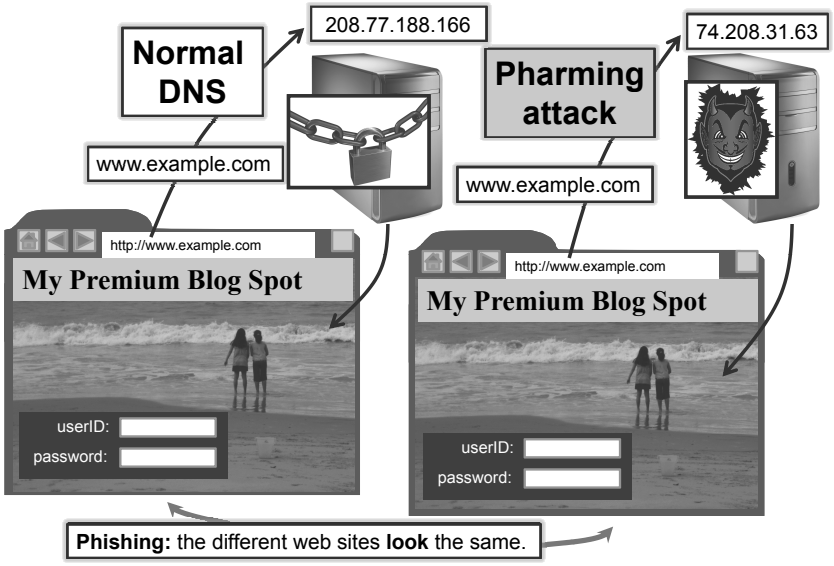

Figure 6.5: A pharming attack that maps a domain name to a malicious server, which then performs a phishing attack by delivering a web page that looks the same as the real one, to trick people into entering their userIDs and passwords. (Image by Karen Goodrich; used with permission.)

Other Pharming Attacks

Victims of a combined pharming and phishing attack would have no way of distinguishing between the fake and real sites, since all of the information conveyed by the browser indicates that they are visiting a trusted web site. (See also Section 7.2.2.) There are other types of pharming attacks, as well.

For instance, email relies on specialized DNS entries known as MX records, so another possible pharming attack allows an attacker to redirect mail intended for certain domains to a malicious server that steals information. Given that many online services allow password recovery through email, this could be a means of performing identity theft.

Other pharming attacks might associate the domain name used for operating system updates with a malicious IP address, causing victims to automatically download and execute malicious code instead of a needed software patch. In fact, the possibilities of damage from pharming attacks are nearly endless because of the large degree of trust placed on the truthfulness of domain-name resolutions. Thus, DNS compromises can have dire consequences for Internet users.

DNS Cache Poisoning

Some DNS attacks are made possible by a technique known as ***DNS cache poisoning***. In this technique, an attacker attempts to trick a DNS server into caching a false DNS record, which will then cause all downstream clients issuing DNS requests to that server to resolve domains to attacker-supplied IP addresses. Consider the following DNS cache poisoning scenario:

1. An attacker, Eve, has decided to launch a DNS cache poisoning attack against an ISP DNS server. She rapidly transmits DNS queries to this server, which in turn queries an authoritative name server on behalf of Eve.

2. Eve simultaneously sends a DNS response to her own query, spoofing the source IP address as originating at the authoritative name server, with the destination IP set to the ISP DNS server target.

3. The ISP server accepts Eve's forged response and caches a DNS entry associating the domain Eve requested with the malicious IP address Eve provided in her forged responses. At this point, any downstream users of that ISP will be directed to Eve's malicious web site when they issue DNS requests to resolve the domain name targeted by Eve.

There are several obstacles an attacker like Eve must overcome to issue a fake DNS response that will be accepted. First, an attacker must issue

a response to her own DNS query before the authoritative name server is given a chance to respond. This obstacle is easily overcome, however, because if the attacker forces the target name server to query external authoritative name servers, she can expect that her immediate, direct response will be received before these external name servers have a chance to perform a lookup and issue a reply. Second, each DNS request is given a 16-bit query ID. If the response to a query is not marked with the same ID as its corresponding request, it will be ignored. In 2002, it was revealed that most major DNS software simply used sequential numbers for query IDs, however, allowing easy prediction and circumvention of this naive authentication. (See Figure 6.6.) Once this bug was disclosed, most DNS software vendors began to implement randomization of query IDs.

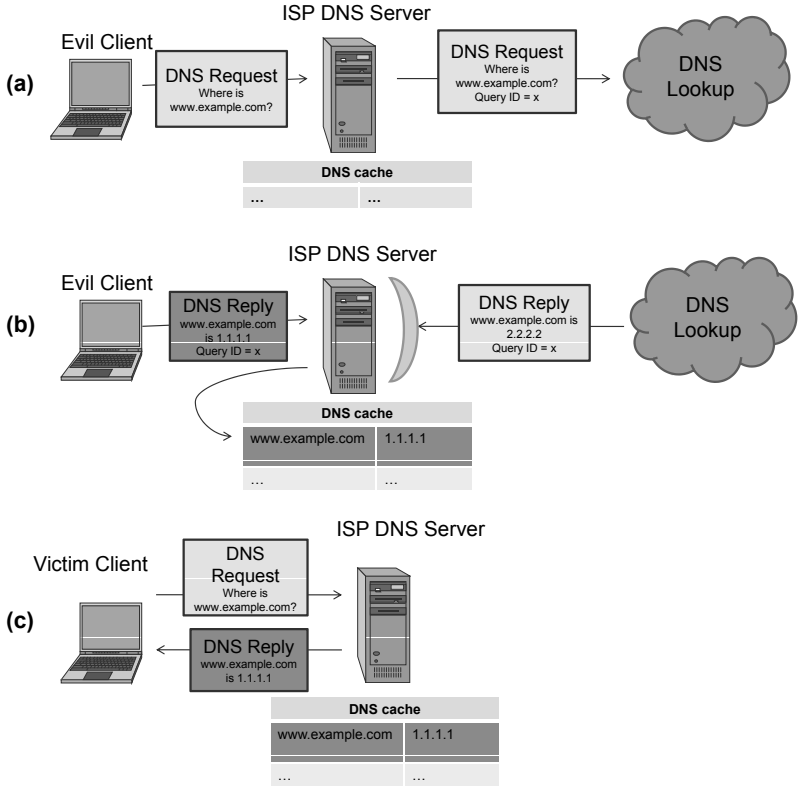

Figure 6.6: A DNS cache poisoning attack: (a) First, the attacker sends a DNS request for the domain he wishes to poison. The ISP DNS server checks its cache and queries root name servers for the domain. (b) The attacker sends a corresponding reply for his own request, guessing the transaction ID. If he successfully guesses the random query ID chosen by the ISP DNS server, the response will be cached. (c) Any clients of the ISP DNS server issuing DNS requests for the poisoned domain will be redirected to the attacker's IP address.

DNS Cache Poisoning and the Birthday Paradox

Unfortunately, randomization of transaction IDs does not completely solve the problem of DNS cache poisoning. If an attacker can successfully guess the ID associated with an outbound DNS request and issue a response with the same ID, the scenario above would still be possible, as depicted in Figure 6.6. This guessing is actually more likely if the attacker issues a lot of fake requests and responses to the same domain name lookup.

This increase in attack success probability from an increase in fake requests is a result of a principle known as the ***birthday paradox***, which states that the probability of two or more people in a group of 23 sharing the same birthday is greater than 50%. This intuitively surprising result is due to the fact that in a group of 23 people, there are actually $23 \cdot 22/2 = 253$ ***pairs*** of birthdays, and it only takes one matching pair for the birthday paradox to hold. (The birthday paradox and its use to find collisions in a hash function is also discussed in Section 8.3.2.)

Let us apply the reasoning of the birthday paradox to DNS cache poisoning. An attacker issuing a fake response will guess a transaction ID equal to one of n different 16-bit real IDs with probability $n/2^{16}$; hence, she would fail to match one with probability $1 - n/2^{16}$. Thus, an attacker issuing n fake responses will fail to guess a transaction ID equal to one of n different 16-bit real IDs with probability

$$\left(1 - \frac{n}{2^{16}}\right)^n.$$

By issuing at least $n = 213$ requests and an equal number of random fake responses, an attacker will have roughly at least a 50% chance that one of her random responses will match a real request. (See Figure 6.7.)

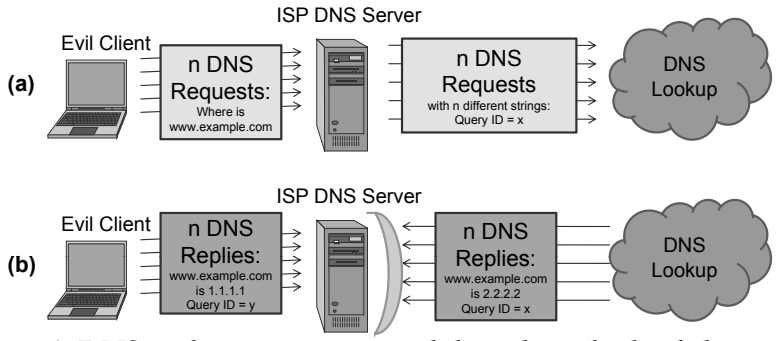

Figure 6.7: A DNS cache poisoning attack based on the birthday paradox: (a) First, an attacker sends n DNS requests for the domain she wishes to poison. (b) The attacker sends n corresponding replies for her own request. If she successfully guesses one of the random query IDs chosen by the ISP DNS server, the response will be cached.

Subdomain DNS Cache Poisoning

Despite the birthday paradox, the above guessing attack is extremely limited because of its narrow time frame. Recall that when a correct response to a DNS query is received, that result is cached by the receiving server and stored for the time specified in the time-to-live field. When a name server has a record in its cache, it uses that record rather than issuing a new query to an authoritative name server. As a result, the attacker can only make as many guesses as he can send in the time between the initial request and the valid reply from the authoritative name server. On each failed guessing attempt, the valid (harmless) response will be cached by the targeted name server, so the attacker must wait for that response to expire before trying again. Responses may be cached for minutes, hours, or even days, so this slowdown makes the attack described above almost completely infeasible.

Unfortunately, a new ***subdomain DNS cache poisoning*** attack was discovered in 2008 that allows attackers to successfully perform DNS cache poisoning by using two new techniques. Rather than issuing a request and response for a target domain like example.com, which would only allow one attempt at a time, the attacker issues many requests, each for a different nonexistent subdomain of the target domain. For example, the attacker might send requests for subdomains aaaa.example.com, aaab.example.com, aaac.example.com, and so on. These subdomains don't actually exist, of course, so the name server for the target domain, example.com, just ignores these requests. Simultaneously, the attacker issues responses for each of these requests, each with a guessed transaction ID. Because the attacker now has so many chances to correctly guess the response ID and there is no competition from the target domain to worry about, it is relatively likely that the attack will be successful. This new attack was shown to be successful against many popular DNS software packages, including BIND, the most commonly used system.

Using Subdomain Resolution for DNS Cache Poisoning

By itself, this attack accomplishes little—on a successful attempt, the attacker only manages to poison the DNS record for a nonexistent domain. This is where the second new technique comes into play. Rather than simply reply with an address for each fake subdomain like abcc.example.com, the attacker's responses include a glue record that resolves the name server of the target domain, example.com, to an attacker-controlled server. Using this strategy, on successfully guessing the transaction ID the attacker can control not just one DNS resolution for a nonexistent domain but all resolutions for the entire target domain.

Client-Side DNS Cache Poisoning Attacks

In addition to attacks on name servers, a similar DNS cache poisoning attack can be conducted against a target client as depicted in Figure 6.8.

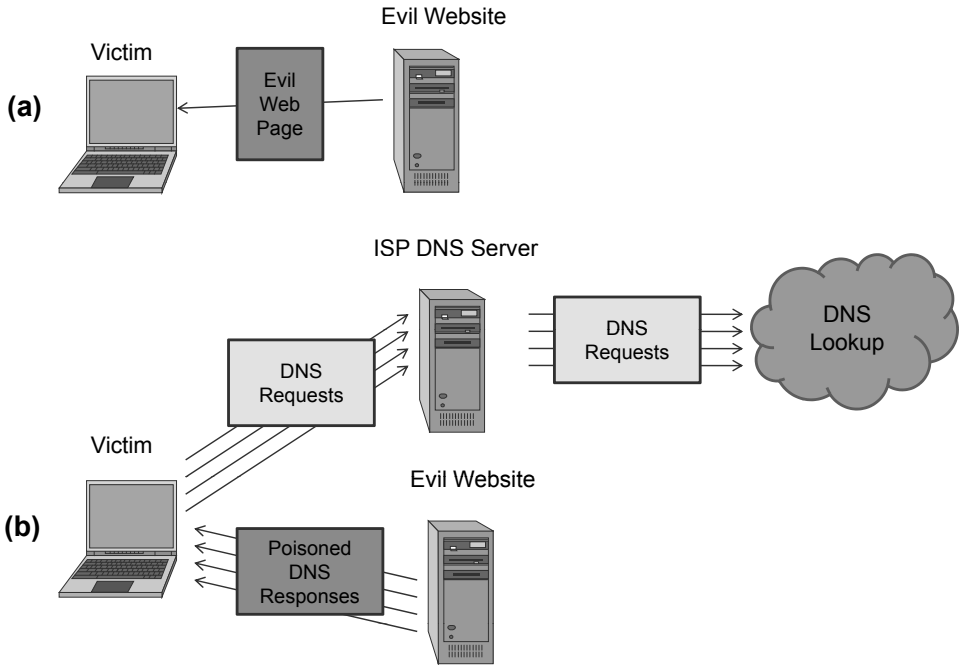

Figure 6.8: A DNS cache poisoning attack against a client: (a) On visiting a malicious web site, the victim views a page containing many images, each causing a separate DNS request to be made to a nonexistent subdomain of the domain that is to be poisoned. (b) The malicious web server sends guessed responses to each of these requests. On a successful guess, the client's DNS cache will be poisoned.

An attacker can construct a malicious web site containing HTML tags that automatically issue requests for additional URLs such as image tags. These image tags each issue a request to a different nonexistent subdomain of the domain the attacker wishes to poison. When the attacker receives indication that the victim has navigated to this page, he can rapidly send DNS replies with poisoned glue records to the client. On a successful attack, the client will cache the poisoned DNS entry.

This type of attack is especially stealthy, since it can be initiated just by someone visiting a web site that contains images that trigger the attack. These images will not be found, of course, but the only warning the user has that this is causing a DNS cache poisoning attack is that the browser window may display some icons for missing images.

Identifying the Risks of Subdomain DNS Cache Poisoning

The subdomain DNS cache poisoning attack does not rely on a vulnerability in a specific implementation of DNS, which could be problematic in its own right. Instead, the attack demonstrates two weakness in the DNS protocol itself:

- Relying on a 16-bit number as the only mechanism for verifying the authenticity of DNS responses, which is insufficient for security
- Having the response for a nonexisting subdomain request be a nonresponse

As such, this form of DNS cache poisoning is difficult to prevent. It would be a daunting task to actually fix the underlying vulnerabilities by forcing the adoption of a new version of DNS, given the critical nature of DNS in the Internet's infrastructure. Instead, several stopgap measures have been put in place to reduce the risk of attack until a more permanent solution is developed.

Some Defenses Against Subdomain DNS Cache Poisoning

First, most DNS cache poisoning attacks are targeted towards ISP DNS servers, known as ***local DNS*** (***LDNS***) servers, rather than authoritative name servers. Prior to more recent cache poisoning attacks, the practice of leaving LDNS servers openly accessible to the outside world was common, but since 2008, most LDNS servers have been reconfigured to only accept requests from within their internal network. This prevents all cache poisoning attempts originating from outside of an ISP's network. However, the possibility of attacking from within the network remains.

To further reduce the chances of a successful attack, many DNS implementations now incorporate ***source-port randomization*** (***SPR***), the practice of randomizing the port from which DNS queries originate (and must be replied to). This decreases the likelihood of successfully generating a false DNS reply that will be accepted. In addition to the 2^{16} possible query IDs, the number of possible combinations is multiplied by the number of possible source ports, which typically numbers around 64,000. While this additional randomness is an improvement, it has been demonstrated that DNS cache poisoning is still possible against name servers using both random query IDs and source-port randomization.

6.1.4 DNSSEC

Since the stopgap measures mentioned above are insufficient to completely mitigate the risk of DNS cache poisoning, a new approach to DNS must be taken. One possible solution is the adoption of ***DNSSEC***, which is a set of security extensions to the DNS protocol that prevent attacks such as cache poisoning by digitally signing all DNS replies using public-key cryptography. Such signatures make it infeasible for an attacker to spoof a DNS reply and thereby poison a DNS cache.

One challenge to the widespread implementation of DNSSEC, however, is that it represents an extension to the DNS protocol itself; hence, in order to work, DNSSEC must be deployed at both the client and server ends. At the time of this writing, DNSSEC is being deployed more and more frequently, but it has yet to be adopted universally. Thus, until it or something like it is widely adopted, there will still be security risks in the DNS protocol.

DNSSEC uses several new types of DNS records. When a client issues a DNS request, the request packet indicates that DNSSEC is supported. If the queried server also supports DNSSEC, then a ***resource-record signature*** (***RRSIG***) record is returned alongside any resolved queries. The RRSIG record contains a digital signature of the returned records computed by generating a hash of the returned records and encrypting this hash with the authoritative name server's private key. In addition to the RRSIG record, the response to the client contains a ***DNSKEY*** record containing the authoritative name server's public key. The client can then verify the authenticity of the returned records by decrypting the digital signature using the name server's public key and comparing the hash to a locally computed hash of the records.

The only step remaining is to establish trust in the name server's supposed public key. This is essential to the security of the system. Otherwise, an attacker could simply intercept traffic, sign fake DNS response records with his own private key, and send his own public key as a DNSKEY record. To prevent this type of attack, DNSSEC employs a ***chain of trust***. Recalling that each DNS zone (besides the root zone) has a parent zone, trust can be established by relying on a hierarchy working back up to the root name server. To validate a particular zone's public key, the client requests a ***designated signer*** (***DS***) record from that zone's parent, which contains a hash of the child zone's public key. In addition to this DS record, the parent name server returns its own DNSKEY record and another RRSIG record containing a digitally signed copy of the DS record.

To perform signature verification, the client uses the parent name server's DNSKEY to decrypt the RRSIG record, compares this to the DS record, and finally compares the DS record to the child name server's DNSKEY. This process is repeated until a "trusted key" that the client has existing knowledge of and does not need to verify is encountered. Ideally, the root name server would represent this point of trust, but at the time of this writing, the root name server does not provide DNSSEC. For now, DNSSEC clients must be configured with other known trust points at levels below the root name server. (See Figure 6.9.)

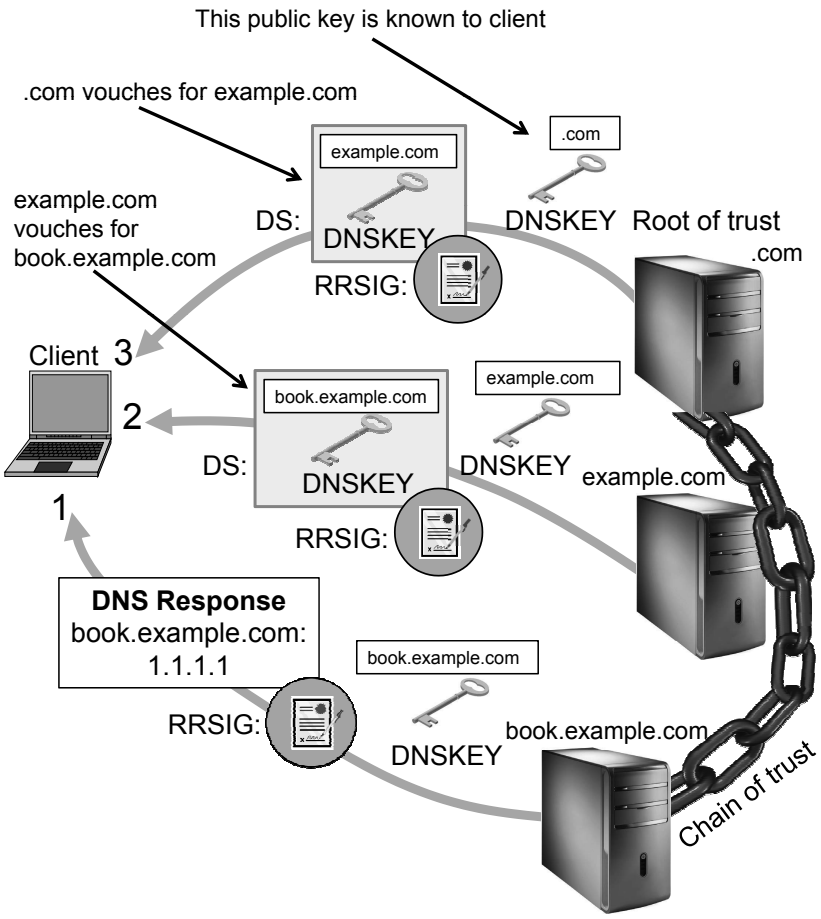

Figure 6.9: A DNSSEC response and the chain of trust that validates it. In this case, book.example.com returns a signed DNS response along with its public key, example.com sends its public key and a signed DS record validating the public key of book.example.com, and .com sends its public key and a signed DS record validating the public key of example.com. The client can trust this chain, since it knows the public key of .com.

6.2 Firewalls

It is now an accepted fact that the Internet is a vast network of untrusted and potentially malicious machines. In order to protect private networks and individual machines from the dangers of the greater Internet, a ***firewall*** can be employed to filter incoming or outgoing traffic based on a predefined set of rules that are are called ***firewall policies***.

Firewalls may be used both as a protective measure, to shield internal network users from malicious attackers on the Internet, or as a means of censorship. For example, many companies prevent internal users from using certain protocols or visiting certain web sites by employing firewall technology. On a much larger scale, some countries, such as China, impose censorship of their citizens by subjecting them to restrictive national firewall policies that prevent users from visiting certain types of web sites.

Firewalls can be implemented in either hardware or software, and are typically deployed at the perimeter of an internal network, at the point where that network connects to the Internet. (See Figure 6.10.) In this model of network topography, the Internet is considered an untrusted zone, the internal network is considered a zone of higher trust, and any machines, like a firewall, situated between the Internet and the internal trusted network are in what is known as a ***demilitarized zone***, or ***DMZ*** (borrowing terminology from the military). Incidentally, firewalls are also commonly implemented in software on personal computers.

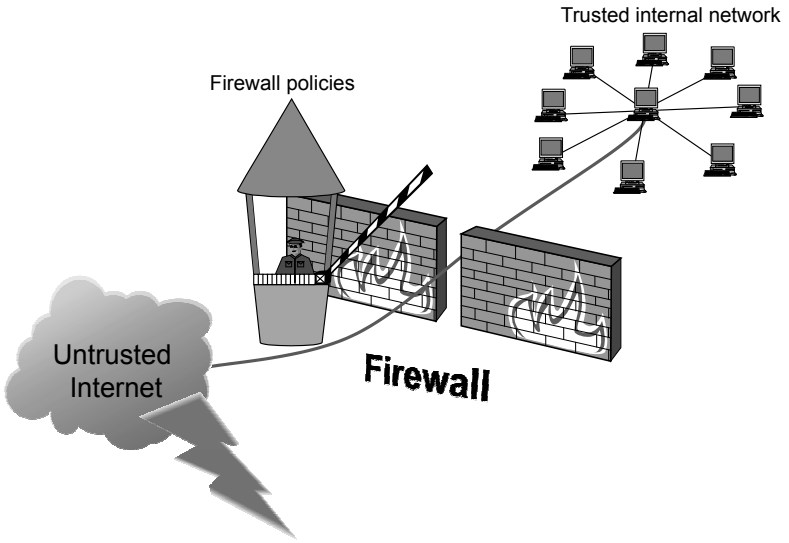

Figure 6.10: A firewall uses firewall policies to regulate communication traffic between the untrusted Internet and a trusted internal network.

6.2.1 Firewall Policies

Before examining the specifics of how firewalls are implemented, it is important to understand the different conceptual approaches to defining firewall policies for an organization or machine. Packets flowing through a firewall can have one of three outcomes:

- ***Accepted***: permitted through the firewall
- ***Dropped***: not allowed through with no indication of failure
- ***Rejected***: not allowed through, accompanied by an attempt to inform the source that the packet was rejected

Policies used by the firewall to handle packets are based on several properties of the packets being inspected, including the protocol used (such as TCP or UDP), the source and destination IP addresses, the source and destination ports, and, in some cases, the application-level payload of the packet (e.g., whether it contains a virus).

Blacklists and White Lists

There are two fundamental approaches to creating firewall policies (or ***rulesets***) to effectively minimize vulnerability to the outside world while maintaining the desired functionality for the machines in the trusted internal network (or individual computer). Some network administrators choose a ***blacklist*** approach, or ***default-allow*** ruleset. In this configuration, all packets are allowed through except those that fit the rules defined specifically in a blacklist. This type of configuration is more flexible in ensuring that service to the internal network is not disrupted by the firewall, but is naive from a security perspective in that it assumes the network administrator can enumerate all of the properties of malicious traffic.

A safer approach to defining a firewall ruleset is to implement a ***white list*** or ***default-deny*** policy, in which packets are dropped or rejected unless they are specifically allowed by the firewall. For example, a network administrator might decide that the only legitimate traffic entering the network is HTTP traffic destined for the web server and that all other inbound traffic should be dropped. While this configuration requires greater familiarity with the protocols used by the internal network, it provides the greatest possible caution in deciding which traffic is acceptable.

6.2.2 Stateless and Stateful Firewalls

Firewalls can support policies that are based on properties of each packet in isolation, or they can consider packets in a broader context.

Stateless Firewalls

One simple implementation of a firewall is known as a ***stateless firewall***.

Such a firewall doesn't maintain any remembered context (or "state") with respect to the packets it is processing. Instead, it treats each packet attempting to travel through it in isolation without considering packets that it has processed previously. In particular, stateless firewalls don't have any memory dedicated to determining if a given packet is part of an existing connection. Stateless firewalls simply inspect packets and apply rules based on the source and destination IP addresses and ports.

While stateless firewalls provide a starting point for managing traffic flow between two untrusted zones and require little overhead, they lack flexibility and often require a choice between limited functionality and lax security. Consider the case of a user on the internal network wishing to connect via TCP to an external web site. First, the user initiates the connection by sending a TCP packet marked with the SYN flag set as discussed in Section 5.4.1. In order for this packet to be allowed, the firewall must permit outbound packets originating at the user's IP from whichever port the user sends the request. Next, the web server responds with a packet that has the SYN and ACK flags set. For this packet to be allowed through, the firewall must allow inbound packets sent from the web server, presumably originating from the appropriate port for web traffic. (See Figure 6.11.)

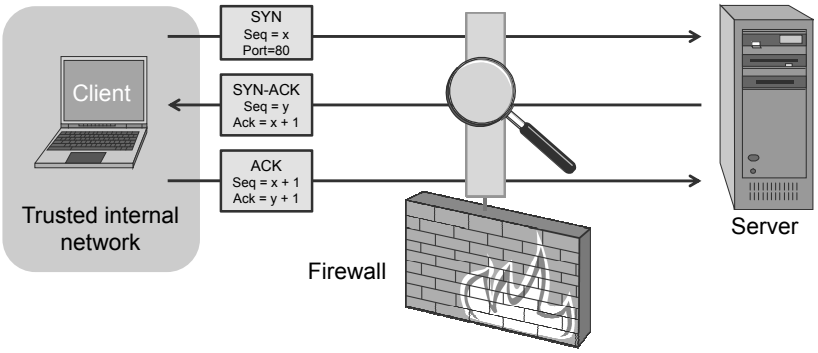

Figure 6.11: A stateless firewall allowing TCP sessions initiating an HTTP connection (port 80) with a request from the trusted internal network.

Blocking Undesired Packets

Note that if the above policy were in place, all traffic from a web server originating at the default port for web servers would be allowed through the firewall to the user's machine, which may be undesirable. This policy can be tightened somewhat by observing that the firewall does not need to allow TCP packets marked with just the SYN flag to reach the user. (See Figure 6.12.) While this restriction would prevent outside parties from initiating TCP connections to an internal machine, it would not prevent them from probing the network with other packets not marked with the SYN flag.

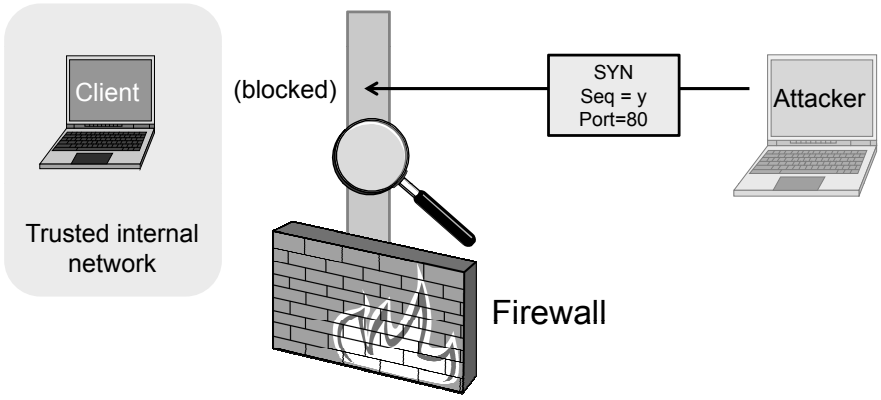

Figure 6.12: A stateless firewall dropping TCP sessions initiating an HTTP connection with a request from outside the trusted internal network.

Stateful Firewalls

Since stateless firewalls don't keep track of any previous traffic, they have no way of knowing whether a particular packet is in response to a previous packet originating within the network or if it is an unprompted packet. ***Stateful firewalls***, on the other hand, can tell when packets are part of legitimate sessions originating within a trusted network. Like NAT devices (Section 5.4.3), stateful firewalls maintain tables containing information on each active connection, including the IP addresses, ports, and sequence numbers of packets. Using these tables, stateful firewalls can solve the problem of only allowing inbound TCP packets that are in response to a connection initiated from within the internal network. Once the initial handshake is complete and allowed through the firewall, all subsequent communication via that connection will be allowed, until the connection is finally terminated. (See Figure 6.13.)

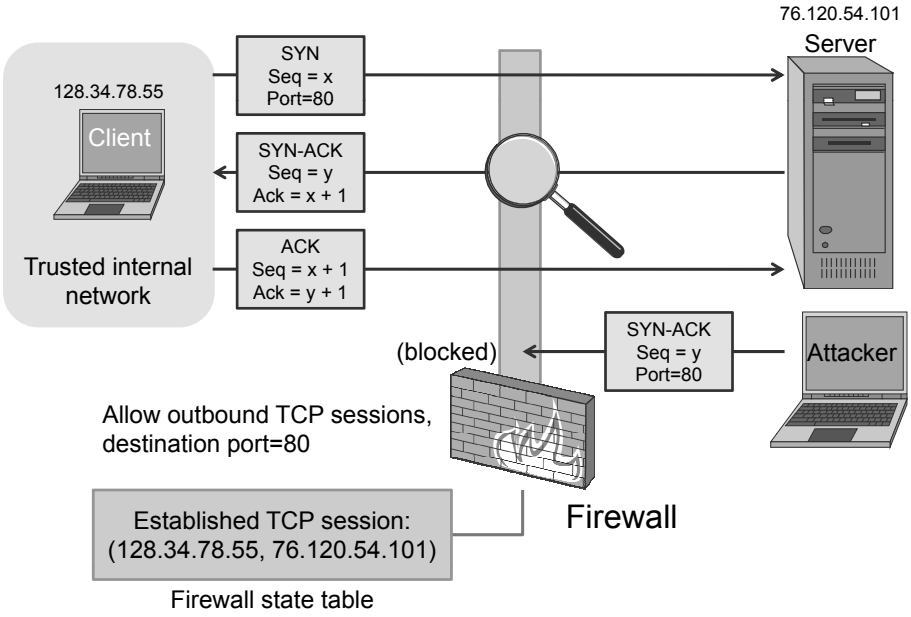

Figure 6.13: A statefull firewall configured to allow TCP web sessions (port 80) with a request coming from inside the trusted internal network.

Handling TCP connections is relatively straightforward because both parties must perform an initial handshake to set up the connection. Handling UDP traffic is not as clear. Most stateful firewalls consider a UDP "session" (an abstraction that is not reflected in the underlying protocol) to be started when a legitimate UDP packet is allowed through the firewall. At this point, all subsequent UDP transmissions between the same two IPs and ports are allowed, until a specified timeout is reached.

Stateful firewalls allow administrators to apply more restrictive rules to network traffic and create more effective policies for inbound versus outbound traffic. However, sometimes it is desirable to be able to manage traffic based on the actual contents of packets entering and exiting a network rather than merely considering the origin and destination. This is possible through the use of ***application-layer firewalls***. As the name indicates, these firewalls are capable of examining the data stored at the application layer of inbound and outbound packets, and apply rules based on these contents. For example, simple rules might reject all requests for a particular web site. Most modern firewalls employ some level of higher-layer filtering, which depends on the properties of an IP packet's payload, such as the properties of the headers of TCP and UDP packets. In general, the practice of examining higher-layer data in network traffic is known as ***deep packet inspection***. It is frequently used in conjunction with intrusion detection systems and intrusion prevention systems to make sophisticated policies delineating acceptable and potentially malicious traffic.

6.3 Tunneling

As we have mentioned, one of the challenges of Internet communication is that it is not secure by default. The contents of TCP packets are not normally encrypted, so if someone is eavesdropping on a TCP connection, he can often see the complete contents of the payloads in this session. One way to prevent such eavesdropping without changing the software performing the communication is to use a ***tunneling*** protocol. In such a protocol, the communication between a client and server is automatically encrypted, so that useful eavesdropping is infeasible. To use such a protocol, the client and server have to have some way of establishing encryption and decryption keys, so using a tunneling protocol requires some setup. Unfortunately, the content of this setup requires the use of application-layer concepts, such as identity and authorization, in transport-layer or network-layer protocols. As a result, tunneling technology allows one to solve some security weaknesses with TCP/IP protocols at the expense of adding overhead to the IP protocol stack. Nevertheless, tunneling is now a widely used technology, since it allows users to communicate securely across the untrusted Internet. (See Figure 6.14.)

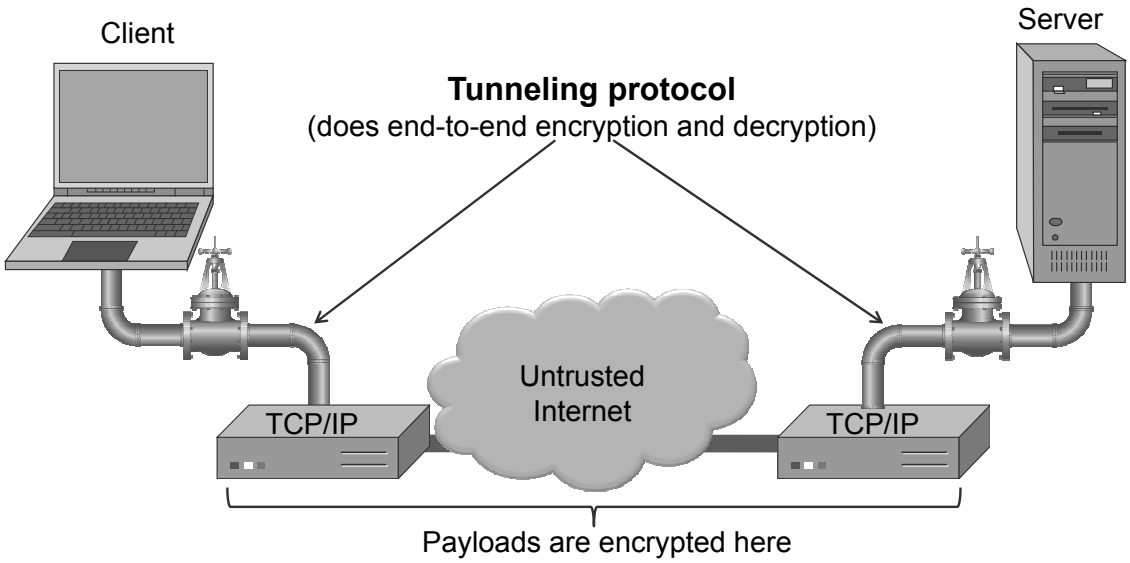

Figure 6.14: Tunneling protocols provide end-to-end encryption of TCP/IP communication between a client and a server.

6.3.1 Secure Shell (SSH)

In the early days of the Internet, it became clear that the ability to administer a machine remotely was a powerful capability. Early remote administration protocols such as telent, FTP, and rlogin allowed administrators to control machines remotely via a command prompt or shell, but provided no form of encryption and instead sent data in plaintext. To remedy these insecure protocols, ***SSH*** was created to use symmetric and public-key cryptography to communicate across the Internet using an encrypted channel.

The security of SSH is based on the combination of the respective strengths of the encryption, decryption, and key exchange algorithms that SSH uses. Because of its strong security, the SSH protocol is used for a variety of tasks in addition to secure remote administration, including file transfer through the simple ***Secure Copy Protocol*** (***SCP***) or as part of the more full-featured ***Secure File-Transfer Protocol*** (***SFTP***).

In addition, one of the most common uses of the ***SSH*** protocol is for secure tunneling. Because the protocol is designed such that an eavesdropper cannot deduce the contents of SSH traffic, a tunnel established using SSH will prevent many attacks based on packet sniffing. To establish an SSH connection, a client and server go through the following steps:

1. The client connects to the server via a TCP session.
2. The client and server exchange information on administrative details, such as supported encryption methods and their protocol version, each choosing a set of protocols that the other supports.
3. The client and server initiate a secret-key exchange to establish a shared secret session key, which is used to encrypt their communication (but not for authentication). This session key is used in conjunction with a chosen block cipher (typically AES, 3DES, Blowfish, or IDEA) to encrypt all further communications.
4. The server sends the client a list of acceptable forms of authentication, which the client will try in sequence. The most common mechanism is to use a password or the following public-key authentication method:
   (a) If public-key authentication is the selected mechanism, the client sends the server its public key.
   (b) The server then checks if this key is stored in its list of authorized keys. If so, the server encrypts a challenge using the client's public key and sends it to the client.
   (c) The client decrypts the challenge with its private key and responds to the server, proving its identity.
5. Once authentication has been successfully completed, the server lets the client access appropriate resources, such as a command prompt.

6.3.2 IPsec

One of the fundamental shortcomings of the Internet Protocol is a lack of built-in security measures to ensure the authenticity and privacy of each IP packet. IP itself has no mechanism for ensuring a particular packet comes from a trusted source, since IP packets merely contain a "source address" field that can be spoofed by anyone. In addition, there is no attempt to encrypt data contained in IP packets to guarantee data privacy. Finally, while the IP header contains a noncryptographic checksum for verifying the integrity of the header, there is no attempt to do the same for the payload. The questions of authentication and privacy are addressed in several upper-layer protocols, such as DNSSEC (Section 6.1.4), SSH (Section 6.3.1), and SSL/TLS (Section 7.1.2), but a more powerful solution at the network layer would guarantee security for all applications. To solve these problems, a protocol suite known as ***Internet Protocol Security*** (***IPsec***) was created. IPsec was created in conjunction with IPv6, but was designed to be backwards-compatible for use with IPv4. Because it operates at the network layer, IPsec is completely transparent to applications. Implementing IPsec requires a modified IP stack, but no changes to network applications are necessary.

IPsec consists of several protocols, each addressing different security needs. Each protocol can operate in one of two modes, ***transport mode*** or ***tunnel mode***. In transport mode, additional IPsec header information is inserted before the data of the original packet, and only the payload of the packet is encrypted or authenticated. In contrast, when using tunnel mode, a new packet is constructed with IPsec header information, and the entire original packet, including its header, is encapsulated as the payload of the new packet. Tunnel mode is commonly used to create ***virtual private networks*** (***VPNs***), which are discussed in Section 6.3.3.

In order to use IPsec extensions, the two parties communicating must first set up a set of ***security associations*** (***SAs***), pieces of information that describe how secure communications are to be conducted between the two parties. SAs contain encryption keys, information on which algorithms are to be used, and other parameters related to communication. SAs are unidirectional, so each party must create an SA for inbound and outbound traffic. Communicating parties store SAs in a ***security association database*** (***SADB***). IPsec provides protection for outgoing packets and verifies or decrypts incoming packets by using a ***security parameter index*** (***SPI***) field stored in the IPsec packet header, along with the destination or source IP address, to index into the SADB and perform actions based on the appropriate SA.

Internet Key Exchange (IKE)

IPsec uses the ***Internet Key Exchange*** (***IKE***) protocol to handle the negotiation of SAs. IKE operates in two stages: first, an initial security association is established to encrypt subsequent IKE communications, and second, this encrypted channel is used to define the SAs for the actual IPsec traffic. To establish the initial SA, a secure-key exchange algorithm is used to establish a shared secret key between the two parties. Once this encrypted channel is established, the parties exchange information to define their SAs, including an encryption algorithm, a hash algorithm, and an authentication method such as preshared keys. Once these SAs have been created, the two parties can communicate using IPsec protocols to provide confidentiality, authentication, and data integrity.

The Authentication Header (AH)

The ***Authentication Header*** (***AH***) protocol is used to authenticate the origin and guarantee the data integrity of IPsec packets. The AH, shown in Figure 6.15, is added to an IPsec packet before the payload, which either contains the original IP payload or the entire encapsulated IP packet, depending on whether the transport or tunnel mode is used.

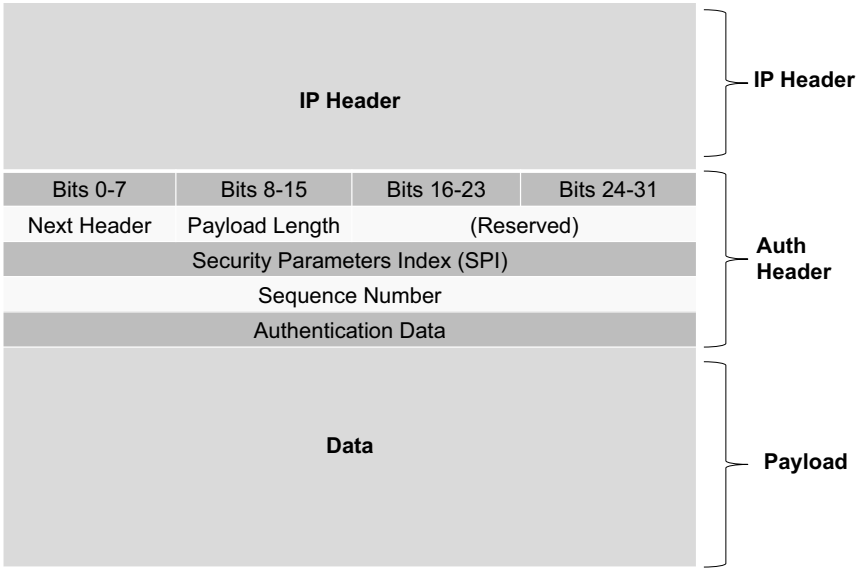

Figure 6.15: The authentication header.

Components of the Authentication Header

The AH header contains a security parameter index (SPI) used to identify the security association associated with the packet, a randomly initialized sequence number to prevent replay attacks, and an "authentication data" field that contains an ***integrity check value*** (***ICV***). The ICV is computed by hashing the entire packet, including the IPsec header, with the exception of fields that may change during routing and the authentication data itself. The hash is computed by using a message authentication code (MAC) (Section 1.3.4), an algorithm that acts as a cryptographic hash function but also makes use of a secret key. The recommended hash function for this MAC is SHA-256. If a malicious party were to tamper with the packet, then the receiving party would discover the discrepancy by recomputing the ICV. In addition, since a secret key is used, only an authenticated party could properly encrypt the payload, verifying the packet's origin. AH's strong authentication comes at a cost. It does not work in conjunction with Network Address Translation (NAT), because its IP source address is included among its authenticated data. Therefore, a NAT device could not successfully rewrite the source IP address while maintaining the ICV of the packet.

The Encapsulating Security Payload (ESP)

Whereas AH provides integrity and origin authentication, it does nothing to guarantee confidentiality—packets are still unencrypted. To satisfy this additional security requirement, the ***encapsulating security payload*** (***ESP***) header, depicted in Figure 6.16, can be used. While AH places a header before the payload or original packet, ESP encapsulates its payload by providing a header and a "trailer." To provide encryption, ESP uses a specified block cipher (typically AES, 3DES, or Blowfish) to encrypt either the entire original IP packet or just its data, depending on whether the tunnel or transport mode is used. ESP also provides optional authentication in the form of an "authentication data" field in the ESP trailer. Unlike AH, ESP authenticates the ESP header and payload, but not the IP header. This provides slightly weaker security in that it does not protect the IP header from tampering, but allows NAT devices to successfully rewrite source IP addresses. However, note that encryption of the payload poses another problem for NAT. Since TCP port numbers are no longer visible to NAT devices, some other identifier must be used to maintain the NAT lookup table.

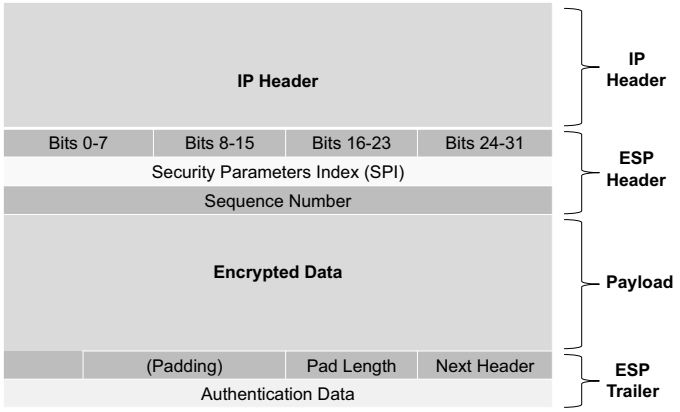

Figure 6.16: The ESP header.

6.3.3 Virtual Private Networking (VPN)

Virtual private networking (***VPN***) is a technology that allows private networks to be safely extended over long physical distances by making use of a public network, such as the Internet, as a means of transport. VPN provides guarantees of data confidentiality, integrity, and authentication, despite the use of an untrusted network for transmission. There are two primary types of VPNs, ***remote access VPN*** and ***site-to-site VPN***.

Remote access VPNs allow authorized clients to access a private network that is referred to as an ***intranet***. For example, an organization may wish to allow employees access to the company network remotely but make it appear as though they are local to their system and even the Internet itself. To accomplish this, the organization sets up a VPN endpoint, known as a ***network access server***, or ***NAS***. Clients typically install VPN client software on their machines, which handle negotiating a connection to the NAS and facilitating communication.

Site-to-site VPN solutions are designed to provide a secure bridge between two or more physically distant networks. Before VPN, organizations wishing to safely bridge their private networks purchased expensive leased lines to directly connect their intranets with cabling. VPN provides the same security but uses the Internet for communication rather than relying on a private physical layer. To create a site-to-site VPN connection, both networks have a separate VPN endpoint, each of which communicates with the other and transmits traffic appropriately.

VPN itself is not standardized, and many companies provide competing VPN solutions. However, most VPN implementations make use of a limited set of protocols to securely transfer data. The details of each of these protocols is beyond the scope of this book, but nearly all use tunneling and

encapsulation techniques to protect network traffic. For example, one of the most widely deployed implementations uses the ***point-to-point tunneling protocol*** (***PPTP***). PPTP works by establishing a connection using the ***peer-to-peer*** (***PPP***) link-layer protocol, then encapsulating PPP frames, which are encrypted using Microsoft Point-to-Point Encryption (MPPE), inside IP packets that can be sent across the Internet. A newer protocol, the ***Layer 2 Tunneling Protocol*** (***L2TP***), was designed to replace PPTP and another older tunneling protocol, Cisco's ***Layer 2 Forwarding*** (***L2F***). The entire L2TP frame, including both header and payload, is encapsulated within a UDP datagram. Within the L2TP packet, a number of link-layer protocols can be encapsulated, including PPP and Ethernet. L2TP is commonly used in conjunction with IPsec to ensure authentication, integrity, and confidentiality.

Some Risks in Allowing for VPNs and Tunneling

While VPNs and other secure tunneling technologies solve one security problem (i.e., how to communicate securely across the Internet), they actually can create another. In particular, one of the most common methods to circumvent firewall policy relies on the use of tunneling. When using a tunneling protocol, the payloads of a series of network packets are encapsulated in a different delivery protocol that might otherwise be blocked by a firewall. Deep packet inspection is useless in this case (other than to detect that a tunnel protocol is being used), since the payloads in a tunnel protocol are encrypted.

For instance, an information-leakage attack, such as sending company secrets out of a compromised network using HTTP packets, becomes more difficult to detect when protocols relying on tunneling are used. Because tunnel protocols are designed such that an eavesdropper cannot deduce the contents of the encrypted traffic, no amount of deep packet inspection can determine whether the tunneling is being used for a legitimate purpose or whether it is being used as a wrapper for a forbidden protocol. As another example of using tunneling to subvert firewall rules, suppose an organization prevents users from visiting certain web sites from within the internal network. If outbound tunnel connections are allowed, then an internal user could establish a tunnel to an external server that routes HTTP traffic to a forbidden web site on behalf of that user, and sends responses back to the user via the same tunnel. Attackers can also use tunneling to circumvent firewall policy for more malicious purposes. Therefore, it is essential that care be taken when defining acceptable traffic policies for users, especially in regards to protocols that could potentially be used for tunneling.

6.4 Intrusion Detection

An ***intrusion detection system*** (***IDS***) is a software or hardware system that is used to detect signs of malicious activity on a network or individual computer. The functions of an IDS are divided between ***IDS sensors***, which collect real-time data about the functioning of network components and computers, and an ***IDS manager***, which receives reports from sensors.

The ***IDS manager*** compiles data from the IDS sensors to determine if an intrusion has occurred. This determination is usually based on a set of ***site policies***, which are sets of rules and statistical conditions that define probable intrusions. If an IDS manager detects an intrusion, then it sounds an ***alarm*** so that system administrators can react to a possible attack. (See Figure 6.17.)

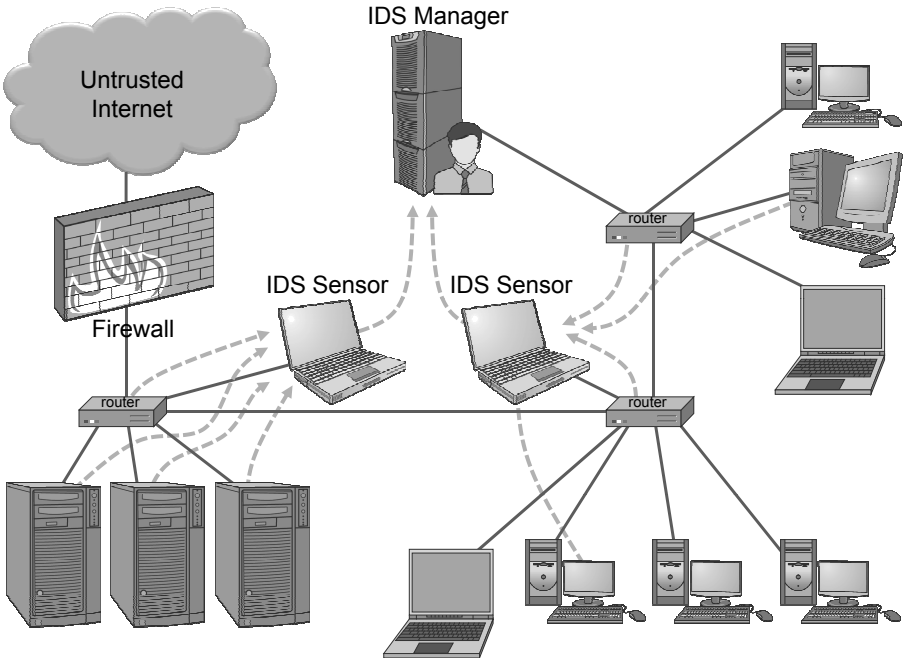

Figure 6.17: A local-area network monitored by an intrusion detection system (IDS). Solid lines depict network connections and gray dashed lines depict data reporting responsibilities. Routers and selected computers report to IDS sensors, which in turn report to the IDS manager.

Intrusions

An IDS is designed to detect a number of threats, including the following:

- ***masquerader***: an attacker who is falsely using the identity and/or credentials of a legitimate user to gain access to a computer system or network
- ***Misfeasor***: a legitimate user who performs actions he is not authorized to do
- ***Clandestine user***: a user who tries to block or cover up his actions by deleting audit files and/or system logs

In addition, an IDS is designed to detect automated attacks and threats, including the following:

- ***port scans***: information gathering intended to determine which ports on a host are open for TCP connections (Section 6.4.4)
- ***Denial-of-service attacks***: network attacks meant to overwhelm a host and shut out legitimate accesses (Section 5.5)
- ***Malware attacks***: replicating malicious software attacks, such as Trojan horses, computer worms, viruses, etc. (Section 4.3)
- ***ARP spoofing***: an attempt to redirect IP traffic in a local-area network (Section 5.2.3)
- ***DNS cache poisoning***: a pharming attack directed at changing a host's DNS cache to create a falsified domain-name/IP-address association (Section 6.1.3)

Intrusion Detection Techniques

Intrusion detection systems can be deployed in a wide variety of contexts to perform different functions. A traditional network intrusion detection system (***NIDS***) sits at the perimeter of a network and detects malicious behavior based on traffic patterns and content. A protocol-based intrusion detection system (***PIDS***) is specifically tailored towards detecting malicious behaviors in a specific protocol, and is usually deployed on a particular network host. For example, a web server might run a PIDS to analyze incoming HTTP traffic and drop requests that may be potentially malicious or contain errors. Similarly, a PIDS may monitor application traffic between two hosts; for example, traffic between a web server and a database might be inspected for malformed database queries. Finally, a host-based IDS

(***HIDS***) resides on a single system and monitors activity on that machine, including system calls, interprocess communication, and patterns in resource usage.

Network IDSs usually work by performing deep packet inspection on incoming and outgoing traffic, and applying a set of attack signatures or heuristics to determine whether traffic patterns indicate malicious behavior. Some network IDSs work by maintaining a database of attack signatures that must be regularly updated, while others rely on statistical analysis to establish a "baseline" of performance on the network, and signal an alert when network traffic deviates from this baseline.

Host IDSs usually work by monitoring audit files and system logs to detect masquerading and misfeasant users who attempt unauthorized actions, and clandestine users who try to delete or modify system monitoring. Such systems typically use heuristic rules or statistical analysis to detect when a user is deviating from "normal" behavior, which could indicate that this user is a masquerading user. Misfeasant users can be detected by a system that has rules defining authorized and unauthorized actions for each user. Finally, clandestine users can be detected by monitoring and logging how changes are made to audit files and system logs themselves.

Passive IDSs log potentially malicious events and alert the network administrator so that action can be taken. They don't take any preemptive actions on their own. On the other hand, more sophisticated reactive systems, known as ***intrusion prevention systems*** (***IPS***), work in conjunction with firewalls and other network devices to mitigate the malicious activity directly. For example, an IPS may detect patterns suggesting a DOS attack, and automatically update the firewall ruleset to drop all traffic from the malicious party's IP address. The most commonly used IPS is an open source solution called Snort, which employs both signature-based detection as well as heuristics.

An IDS Attack

One technique to evade detection is to attempt to launch a denial-of-service attack on the IDS itself. By deliberately triggering a high number of intrusion alerts, an attacker may overwhelm an IDS to the point that it cannot log every event, or at the very least, make it difficult to identify which logged event represents an actual attack and which were used as a diversion. More advanced techniques to evade detection force IDS developers to employ sophisticated heuristics and signature schemes based on state-of-the-art machine learning and artificial intelligence research.

6.4.1 Intrusion Detection Events

Intrusion detection is not an exact science. Two types of errors may occur:

- ***False positive***: when an alarm is sounded on benign activity, which is not an intrusion
- ***False negative***: when an alarm is not sounded on a malicious event, which is an intrusion

Of these two, false negatives are generally considered more problematic because system damage may be going unnoticed. False positives, on the other hand, are more annoying, since they tend to waste time and resources on perceived threats that are not actual attacks. The ideal conditions, then, are as follows. (See Figure 6.18.)

- ***True positive***: when an alarm is sounded on a malicious event, which is an intrusion
- ***True negative***: when an alarm is not sounded on benign activity, which is not an intrusion

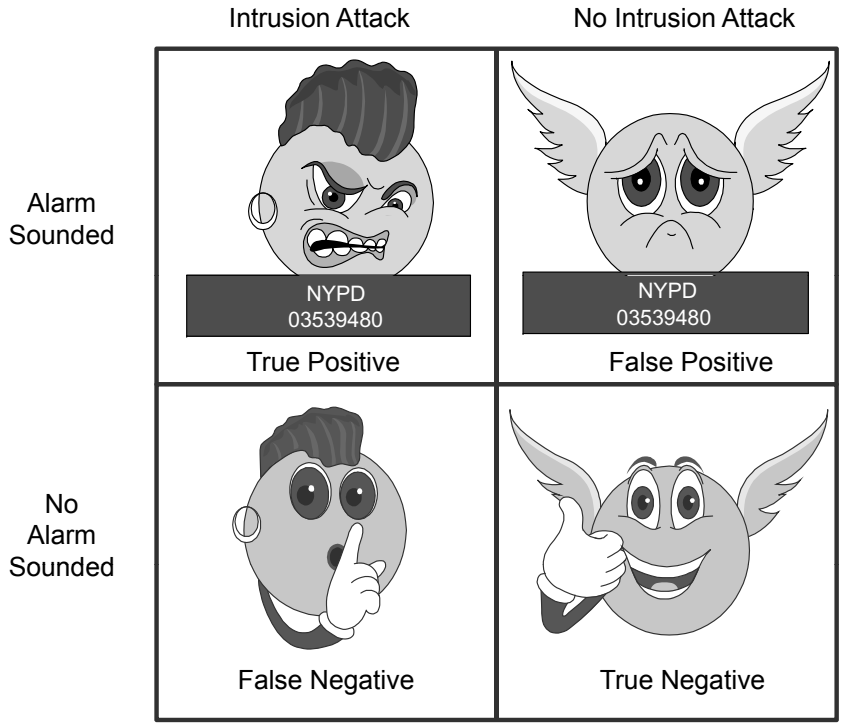

Figure 6.18: The four conditions for alarm sounding by an intrusion detection system.

The Base-Rate Fallacy

Unfortunately, it is difficult to create an intrusion detection system with the desirable properties of having both a high true-positive rate and a low false-negative rate. Often, there are a small number of false positives and false negatives that an intrusion detection system may allow.

If the number of actual intrusions is relatively small compared to the amount of data being analyzed, then the effectiveness of an intrusion detection system can be reduced. In particular, the effectiveness of some IDSs can be misinterpreted due to a statistical error known as the ***base-rate fallacy***. This type of error occurs when the probability of some conditional event is assessed without considering the "base rate" of that event.

This principle can be best illustrated in the context of intrusion detection with an example. Suppose an IDS generates audit logs for system events. Also suppose that when the IDS examines an audit log that indicates real malicious activity (a true positive), it detects the event with probability 99%. This is a high success rate for an IDS, but it still implies that when the IDS examines a benign audit log, it may mistakenly identify a harmless event in that audit log as malicious with a probability of 1% (which would be a false positive).

The base-rate fallacy might convince an administrator that the false-alarm rate is 1%, because that is the rate of failure for the IDS. This is not the case, however. Consider the following scenario:

- Suppose an intrusion detection system generates 1,000,100 audit logs entries.
- Suppose further that only 100 of the 1,000,100 entries correspond to actual malicious events.
- Because of the success rate of the IDS, of the 100 malicious events, 99 will be detected as malicious, which is good.
- Nevertheless, of the 1,000,000 benign events, 10,000 will be mistakenly identified as malicious.
- Thus, there will be 10,099 alarms sounded, 10,000 of which are false alarms, yielding a false alarm rate of about 99%!

Note, therefore, that in order to achieve any sort of reasonable reliability with such an intrusion detection system, the false-positive rate will need to be prohibitively low, depending on the relative number of benign events; hence, care should be taken to avoid the base-rate fallacy when analyzing the probability of misdiagnosing IDS events.

IDS Data Collection and Audit Records

The input to an intrusion detection system is a stream of records that identifies elementary actions for a network or host. The types of actions that are present in such a stream could, for instance, include each HTTP session attempted, each login attempted, and each TCP session initiated for a network-based IDS, and each read, write, or execute performed on any file for a host-based IDS. IDS sensors detect such actions, create records that characterize them, and then either report such records immediately to the IDS manager or write them to an audit log.

In an influential 1987 paper, Dorothy Denning identified several fields that should be included in such event records:

- ***Subject***: the initiator of an action on the target
- ***Object***: the resource being targeted, such as a file, command, device, or network protocol
- ***Action***: the operation being performed by the subject towards the object
- ***Exception-condition***: any error message or exception condition that was raised by this action
- ***Resource-usage***: quantitative items that were expended by the system performing or responding to this action
- ***Time-stamp***: a unique identifier for the moment in time when this action was initiated

For example, if a user, Alice, writes 104 kilobytes of data to a file, dog.exe, then an audit record of this event could look like the following:

[Alice, dog.exe, write, `"no error"`, 104KB, 20100304113451]

Likewise, if a client, 128.72.201.120, attempts to initiate an HTTP session with a server, 201.33.42.108, then an audit record of this event might look like the following:

[128.72.201.120, 201.33.42.108, HTTP, 0.02 CPU sec, 20100304114022]

The exact format for such records would be determined by the IDS designer, and may include other fields as well, but the essential fields listed above should be included.

6.4.2 Rule-Based Intrusion Detection

A technique used by intrusion detection systems to identify events that should trigger alarms is to use ***rules***. These rules could identify the types of actions that match certain known profiles for an intrusion attack, in which case the rule would encode a ***signature*** for such an attack. Thus, if the IDS manager sees an event that matches the signature for such a rule, it would immediately sound an alarm, possibly even indicating the particular type of attack that is suspected.

IDS rules can also encode policies that system administrators have set up for users and/or hosts. If such a rule is triggered, then, by policy, it means that a user is acting in a suspicious manner or that a host is being accessed in a suspicious way. Examples of such policies could include the following:

- Desktop computers may not be used as HTTP servers.
- HTTP servers may not accept (unencrypted) telnet or FTP sessions.
- Users should not read personal directories of other users.
- Users may not write to files owned by other users.
- Users may only use licensed software on one machine at a time.
- Users must use authorized VPN software to access their desktop computers remotely.
- Users may not use the administrative computer server between the hours of midnight and 4:00 am.

Rule-based intrusion detection can be a powerful tool to detect malicious behavior, because each rule identifies an action that policy makers have thought about and have identified as clearly being suspicious. Thus, the potential for annoying false-positive alarms is low, because the policy makers themselves have determined the list of rules. Each rule is there for a reason—if administrators don't like a particular rule, they can remove it, and if they feel that a rule is currently missing, they can add it.

Nevertheless, there are some limitations that may allow knowledgeable attackers to evade rule-based intrusion detection. In particular, signature-based schemes are fundamentally limited in that they require the IDS to have a signature for each type of attack. By performing attacks that might not have a corresponding signature, or by obfuscating the payload of packets containing malicious traffic, signature-based solutions may be bypassed.

6.4.3 Statistical Intrusion Detection

One of the main approaches to intrusion detection is based on statistics. The process begins by gathering audit data about a certain user or host, to determine baseline numerical values about the actions that that person or machine performs. The actions can be grouped by object (that is, all actions having the same object field), action, or exception-condition. Actions could also be aggregated over various time ranges or in terms of ranges or percentages of resource usages. Numerical values that can be derived include:

- ***Count***: the number of occurrences of a certain type of action in the given time range
- ***Average***: the average number of occurrences of a certain type of action in a given of time ranges
- ***Percentage***: the percent of a resource that a certain type of action takes over a given time range
- ***Metering***: aggregates or average-of-averages accumulated over a relatively long period of time
- ***Time-interval length***: the amount of time that passes between instances of an action of a certain type

For example, a system might track how many times a user uses the login program each day, how often a user initiates HTTP sessions, and the typical time interval between times when a user checks his or her email account for new mail. Each of these statistics is gathered and then fed into an artificial-intelligence machine learning system to determine a typical ***profile*** for each user and/or host that the IDS is monitoring.

The profile is a statistical representation of the typical ways that a user acts or a host is used; hence, it can be used to determine when a user or host is acting in highly unusual, anomalous ways. Once a user profile is in place, the IDS manager can determine thresholds for anomalous behaviors and then sound an alarm any time a user or host deviates significantly from the stored profile for that person or machine. (See Figure 6.19.)

Statistical intrusion detection doesn't require any prior knowledge of established intrusion attacks and it has a potential ability to detect novel kinds of intrusions. Since statistical IDSs rely on analyzing patterns in network traffic, it would be difficult for an attacker to hide his behavior from an IDS manager using such techniques. For example, a statistical IDS could learn that a certain user is always out of the office (and not using her computer) on Fridays. Thus, if a login attempt is made on her computer on a Friday, it could be an indication of an intrusion. Likewise, a statistical IDS could learn that a certain network server almost never initiates or

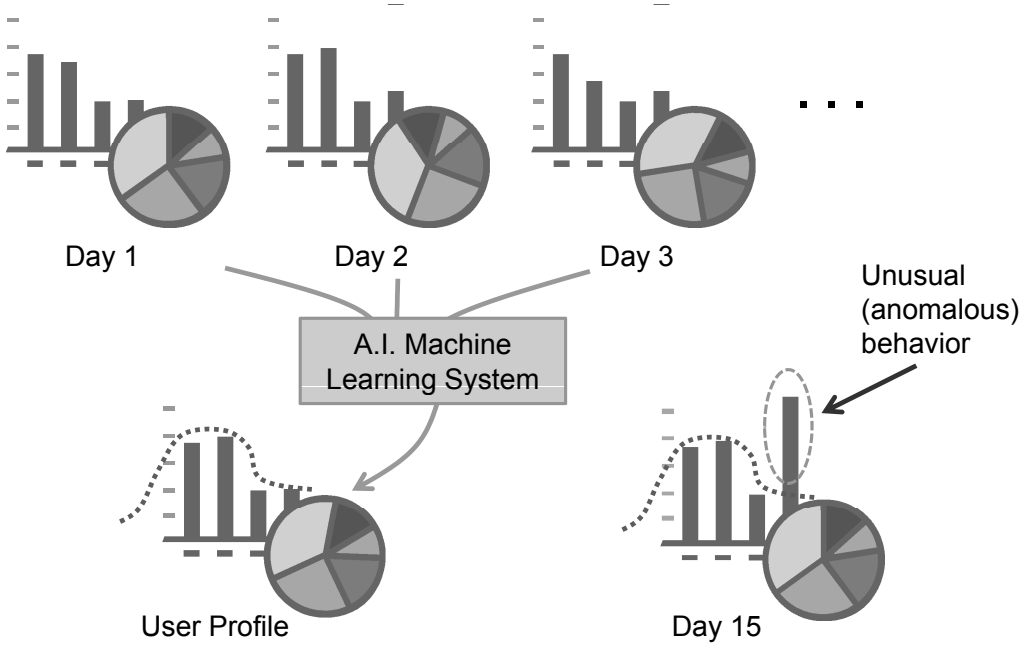

Figure 6.19: How a statistical intrusion detection system works. Statistics about a user are gathered over a sequence of days, and a user profile is determined based on typical behaviors, as defined by an artificial-intelligence machine learning system. Then, on a day when one of the measures is highly unusual compared to the user profile, an alarm would be sounded.

accepts UDP sessions. Thus, an attempt to initiate a UDP connection to this computer could be an attempt to perform some kind of attack.

The potential weakness of statistical methods, however, is that some nonmalicious behavior may generate a significant anomaly, which could lead to the IDS triggering an alarm. Such sensitivity to normal changes in system or user behavior therefore leads to false positives. For example, if a user has an upcoming deadline and suddenly decides to use a new program a large number of times, this might trigger a false alarm. Likewise, if a web server posts some popular content, like a study guide for an upcoming exam, then its usage might exhibit benign behavior that is also anomalous.

In addition, a stealthy attacker may not generate a lot of traffic and thereby might go unnoticed by a statistical network IDS, leading to false negatives. For example, attackers may encapsulate malicious content in benign network protocols such as HTTP, hoping that this traffic will be ignored as ordinary network behavior. Thus, in practice, most intrusion detection systems incorporate both rule-based and statistical methods.

6.4.4 Port Scanning

Determining which traffic is permitted through a firewall and which ports on a target machine are running remote services is a crucial step in analyzing a network for security weaknesses. Any technique that allows a user to enumerate which ports on a machine are accepting connections is known as ***port scanning***. Ports may either be open (accepting connections), closed (not accepting connections), or blocked (if a firewall or other device is preventing traffic from ever reaching the destination port).

Port scanning has a somewhat controversial legal and ethical standing: while it may be used for legitimate purposes to evaluate the security of one's own network, it is also commonly used to perform network reconnaissance in preparation for an attack. Thus, detecting port scanning is a form of preliminary intrusion detection. One of the most popular port scanners in use is nmap, which is available for both Linux and Windows. An example nmap scan is depicted in Figure 6.20.

```
root:~# nmap -sS -O 192.168.1.101

Starting Nmap 4.76 ( http://nmap.org ) at 2009-10-12 15:13 EDT
Interesting ports on 192.168.1.101:
Not shown: 995 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
5001/tcp open  commplex-link
8009/tcp open  ajp13
8180/tcp open  unknown
8888/tcp open  sun-answerbook
Device type: general purpose
Running: Linux 2.6.X
OS details: Linux 2.6.17 - 2.6.25
Network Distance: 0 hops

OS detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1.68 seconds
```

Figure 6.20: Performing a SYN scan with nmap.

Open ports represent a point of contact between the Internet and the application that is listening on that particular port. As such, open ports are potential targets for attack. If a malicious party can successfully exploit a vulnerability in the host operating system or the application listening on an open port, they may be able to gain access to the target system and gain a foothold in the network that could be used for further exploitation. Because of this risk, it is advisable to only open ports for essential network services, and to ensure that the applications listening on these ports are kept up-to-date and patched against recent software vulnerabilities. Likewise, administrators sometimes perform port scans on their own computer networks to reveal any vulnerabilities that should be closed.

As an example of the potential for exploitation, in 2003 a vulnerability was discovered in a Windows remote service known as DCOM–RPC (Distributed Componenet Object Model–Remote Procedural Call). An attacker could craft an exploit that caused a buffer overflow condition in this service, allowing remote code execution and complete control of the target machine. Preventing access to the port this service was running on would prevent successful exploitation.

TCP Scans

There are several techniques for determining the state of the ports on a particular machine. The simplest method of port scanning is known as a ***TCP scan*** or ***connect scan***, in which the party performing the scan attempts to initiate a TCP connection on each of the ports on a target machine. These attempts are done using a standard operating system call for opening a TCP connection at a specified port. Those ports that complete the connection are open, while those that don't are either closed or blocked.

SYN Scans

Another common method is known as a ***SYN scan***, in which the party performing the scan issues a low-level TCP packet marked with the SYN flag for each port on the target machine. If the port is open, then the service listening on that port will return a packet marked with the SYN-ACK flag, and if not, no response will be issued. On receiving a SYN-ACK packet, the scanner issues a RST packet to terminate rather than complete the TCP handshake.

Idle Scanning

One other scanning technique, known as ***idle scanning***, relies on finding a third-party machine, known as a "zombie," that has predictable TCP sequence numbers (Section 5.4.4). The attacker can use the zombie's weak TCP implementation as a tool to perform a port scan on a separate target without leaving any evidence on the target's network. First, the attacker sends a probe, in the form of a SYN-ACK TCP packet, to the zombie. Since this packet was unprompted by the zombie, it will reply to the attacker with a RST packet containing a sequence number. The attacker then sends a SYN packet to the target he wishes to scan, but spoofs the source IP address with that of the zombie machine. If the scanned port is open, the target will reply to the zombie with a SYN-ACK packet. Since the zombie did not open the connection with a SYN packet, it replies to the target with another RST

packet, and increments its sequence number counter. When the attacker probes the zombie again, it checks the received sequence number. If it has been incremented, then the chosen port on the target is open, and if not, the port is either closed or blocked. This process is depicted in Figure 6.21. Since finding a zombie with predictable TCP sequence numbers may be difficult, this scan is not often used in practice, but it provides an effective way to scan a target without leaving any record of the attacker's IP address on the target's network.

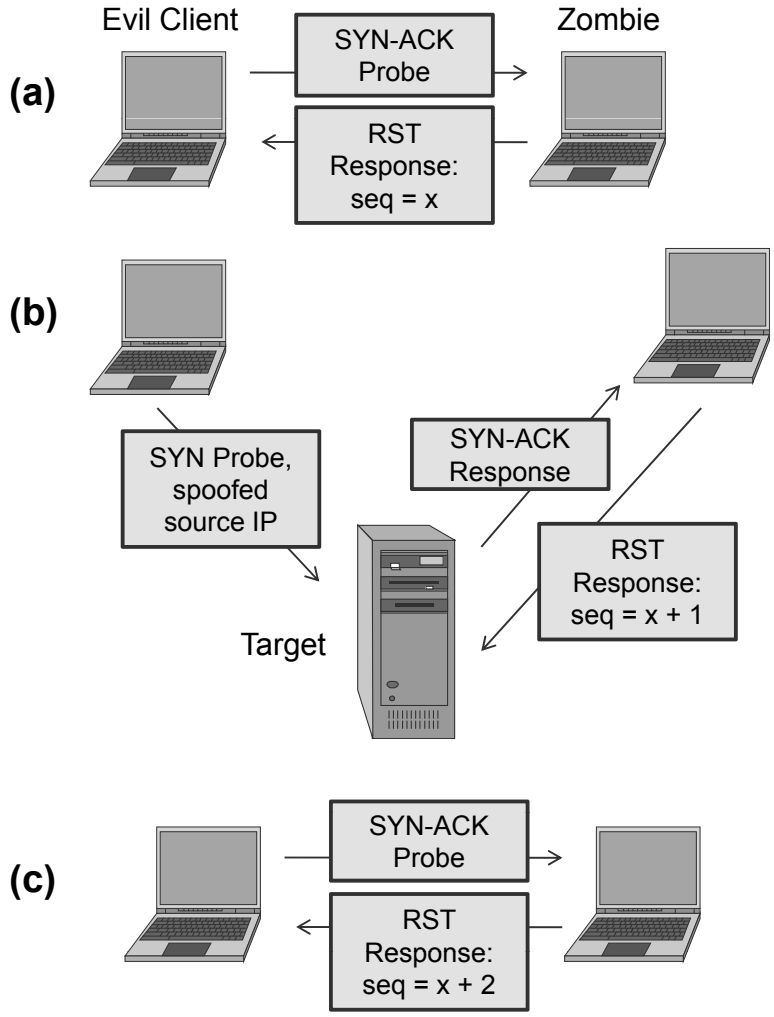

Figure 6.21: An idle scan: (a) The attacker probes a zombie with predictable sequence numbers. (b) The attacker sends a spoofed TCP packet to the target. (c) The attacker checks the state of the port by probing the zombie again.

UDP Scans

While these scans can gather information on TCP ports, a different technique must be used to check the status of UDP ports. Because UDP is a connectionless protocol, there are fewer cues from which to gather information. Most UDP port scans simply send a UDP packet to the specified port. If the port is closed, the target will usually send an ICMP "destination unreachable" packet. If the port is open, then no response will be sent. This scan is not very reliable, however, because open ports and ports blocked by a firewall will both result in no response. To improve the reliability of the response, many port scanners choose to query UDP ports using UDP packets containing the payloads for appropriate applications. For example, to check the status of port 53, the default port for DNS, a port scanner might send a DNS request to the target. This technique may be more reliable, but it is less versatile in that it requires a specialized probe for each target port.

Port Scan Security Concerns

In addition to determining whether ports are open, closed, or blocked, it is often desirable to gain additional information about a target system. In particular, the type and version of each remote service and the operating system version may be valuable in planning an attack. To accomplish this, port scanners may exploit the fact that each operating system has slight differences in its TCP/IP stack implementation and, as such, might respond differently to various requests or probes. Similarly, different implementations and versions of remote services may have subtle differences in the way they respond to certain requests, and knowledge of these differences may allow port scanners to determine the specific service running. This process, known as ***fingerprinting***, is a valuable component of network reconnaissance.

In the early days of port scanning, detecting port scans was simple, since scans would normally proceed sequentially through all possible port numbers. Such scans were then replaced by probing random port numbers, which made detection more difficult but not impossible. For example, a signature for a random port scan could be a sequence of connection attempts made to different destination ports all from the same source IP address. An IDS sensor configured with this signature would be able to alert an IDS manager to a port scan from outside the network. Other port scan detection rules can be defined by noting TCP connection attempts to ports that are known to be closed, as well as port scan detection rules that can be derived from the unique natures of the types of scans previously discussed.

6.4.5 Honeypots

Another tool that can be used to detect intrusions, including port scans, is a ***honeypot***. This is a computer that is used as "bait" for intruders. It is often placed on network in a way that makes it attractive, such as having it configured with software with known vulnerabilities and having its hard drive full of documents that appear to contain company secrets or other apparently valuable information. (See Figure 6.22.)

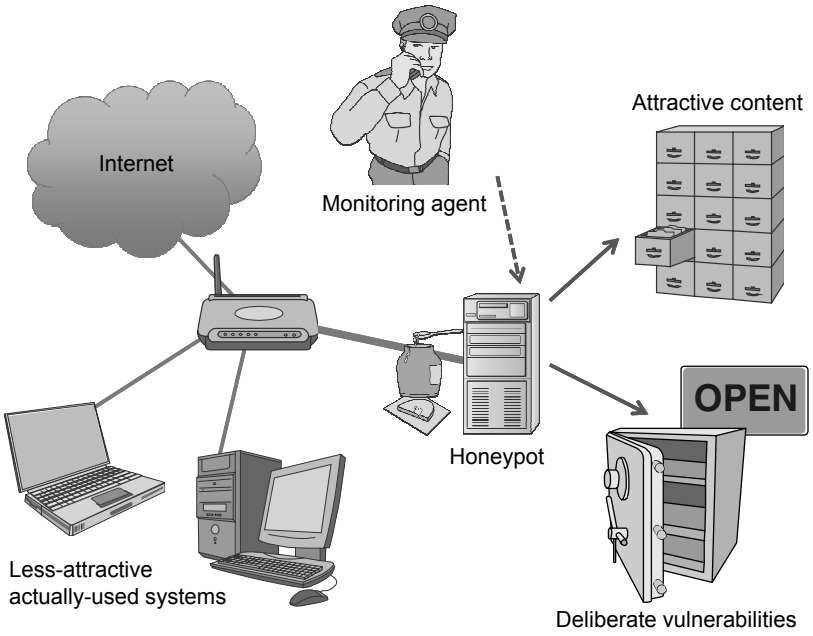

Figure 6.22: A honeypot computer used for intrusion detection.

A honeypot computer is an effective tool for the following reasons:

- ***Intrusion detection***. Since attempts to connect to a honeypot would not come from legitimate users, any connections to a honeypot can be safely identified as intrusions. Based on the way in which such connections are initiated, an intrusion detection system can be updated with the latest attack signatures.
- ***Evidence***. Appealing documents on a honeypot computer encourage an intruder to linger and leave evidence that can possibly lead to the identification of the intruder and/or his location.
- ***Diversion***. A honeypot also may appear to be more attractive to potential intruders than legitimate machines, distracting intruders from sensitive information and services.

6.5 Wireless Networking

The Internet was originally conceived as a means for trusted parties to communicate over a wired network. The advent of wireless networking, however, has introduced many new challenges in providing security to users who may be wirelessly transmitting information that may include untrusted parties. Such challenges include the following. (See Figure 6.23.)

- ***Packet sniffers***. It is much easier to perform packet sniffing in a wireless network, since all the computers sharing a wireless access point are on the same network segment.
- ***Session hijacking***. It is much easier to perform session hijacking, since a computer with a wireless adapter can sniff packets and mimic a wireless access point.
- ***Interloping***. A novel concern in wireless networking is an unauthorized user who is connecting to the Internet through someone else's wireless access point.
- ***Legitimate users***. It is no longer possible to authenticate a legitimate host simply by its physical presence on the local-area network; additional methods for authentication and authorization are needed.

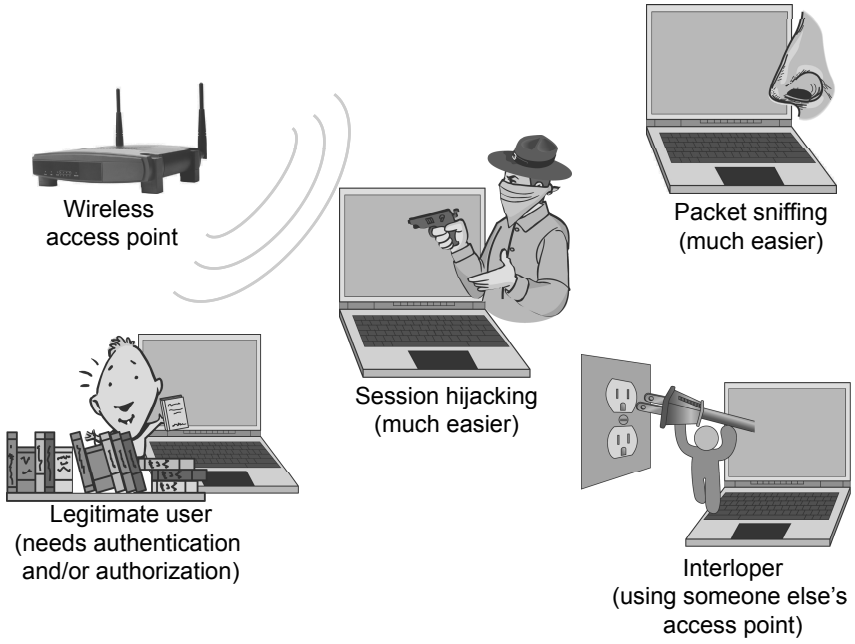

Figure 6.23: Security concerns in wireless networking.

6.5.1 Wireless Technologies

As with all Internet traffic, wireless communications on the Internet make use of the layered IP stack. In wireless networking, parties connecting to a network are referred to simply as ***clients***, while a wireless router or other network interface that a client connects to is known as an ***access point*** (***AP***).

Instead of relying on the Ethernet protocol at the physical and link layers, most wireless networks rely on the protocols defined by the IEEE ***802.11*** family of standards, which define methods for transmitting data via radio waves over predefined radio frequency ranges. In particular, 802.11 defines the structure of wireless frames that encapsulate the higher layers of the IP stack. To allow greater flexibility in handling both wired and wireless data, most TCP/IP implementations perform reframing of packets depending on their intended recipient. For example, wireless traffic received in the form of 802.11 frames is converted into Ethernet frames that are passed to higher layers of the TCP/IP stack. Conversely, Ethernet frames to be routed to wireless clients are converted into 802.11 frames.

Wireless Networking Frames

There are several different frame types defined in the 802.11 standard. First, an ***authentication frame*** is used by a client to present its identity to an access point. If this identity is accepted by the access point, it replies with another authentication frame indicating success. Next, a client sends an ***association request frame***, which allows the access point to allocate resources and synchronize with the client. Again, if the client's credentials are accepted, the access point replies with an ***association response frame***.

To terminate a wireless connection, an access point sends a ***disassociation frame*** to cut off the association, and a ***deauthentication frame*** to cut off communications altogether. If at any point during communications a client becomes accidentally disassociated from the desired access point (if, for example, the client moves to within range of a stronger wireless signal), it may send a ***reassociation request frame***, which will prompt a ***reassociation response frame***. These frames are collectively known as ***management frames*** because they allow clients to establish and maintain communications with access points.

There are three additional common management frames that allow clients and networks to request and broadcast their statuses. In particular, access points can periodically broadcast a ***beacon frame***, which announces its presence and conveys additional information to all clients within range.

In addition to these management frames which set up and maintain communications, ***data frames*** encapsulate the higher levels of the IP stack, and include content from web pages, file transfers, and so on.

6.5.2 Wired Equivalent Privacy (WEP)

Because wireless networks communicate via radio waves, eavesdropping is much easier than with wired networks. In an eavesdropping scenario on a wired network, an attacker must gain access to a physical network interface on the LAN, but when communications are wireless, anyone with appropriate equipment (including most wireless cards) can capture and inspect traffic being sent over the air. The ***Wired Equivalent Privacy*** (***WEP***) protocol was incorporated into the original 802.11 standard with the goal of providing confidentiality, integrity, and access control to a wireless LAN.

WEP Encryption

WEP encrypts each data frame using a ***stream cipher***, which is a symmetric cryptosystem where the ciphertext C is obtained as the exclusive OR of the plaintext message M and a pseudo-random binary vector S generated from the secret key, called ***keystream***:

$$C = M \oplus S.$$

The essence of a stream cipher is the method for generating a keystream of arbitrary length from the secret key, which serves as a seed. (See Figure 6.24.) For a stream cipher to be secure, the same keystream should never be reused or else the attacker can obtain the exclusive OR of two plaintext messages, which enables a statistical attack to recover both the plaintext and the keystream.

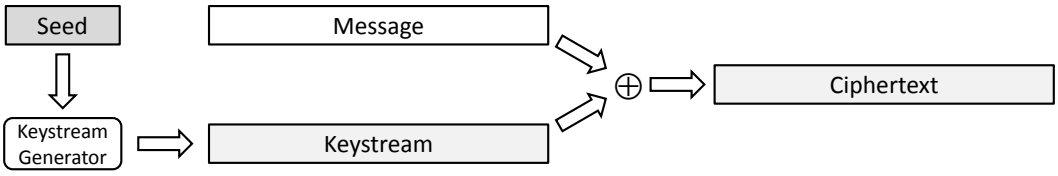

Figure 6.24: Encryption with a stream cipher.

WEP uses the ***RC4*** stream cipher, which is simple and computationally efficient and supports a seed with up to 256 bits. The seed is obtained by concatenating a 24-bit ***initialization vector*** (***IV***) with the ***WEP key***, a secret key that is shared by the client and the access point. In the first version

of WEP, the WEP key had 40 bits, resulting in a 64-bit RC4 seed. Later versions of the protocol extended the key length, resulting in a seed of 128 bits, and eventually 256 bits. To allow decryption, the IV is transmitted together with the ciphertext. The access point then concatenates the IV with the WEP key, generates its own keystream, and computes the message as the exclusive OR of the ciphertext with the keystream. In order to prevent reuse of keystreams, IV values should not be reused. However, the WEP standard does not require access points to check for and reject reused IVs, a vulnerability exploited in several attacks.

For integrity protection, WEP augments the original message with a ***CRC-32 checksum***, which is the output value of a hash function applied to the message. Since CRC-32 is not a cryptographic hash function, it protects the integrity of the message only against transmission errors. Some attacks on WEP exploit this weakness of CRC-32.

WEP Authentication Methods

WEP can be used with two basic authentication methods, ***open system*** and ***shared key*** authentication. When using open system authentication, the client does not need to provide any credentials and can associate itself with the access point immediately. At this point, the client can only send and receive information from the access point using the correct encryption key—if the correct key is not used, the access point ignores the client's requests. In contrast, shared key authentication requires the client to prove possession of the WEP key to the access point before associating with the access point. The access point sends a plaintext challenge to the client, who encrypts it with and sends the ciphertext to the access point. If the received ciphertext decrypts correctly to the challenge, then the client is allowed to associate with the access point.

Attacks on WEP

Intuitively, it may seem as shared key authentication provides stronger security, but in reality this is not the case. Because the challenge is sent to the client in plaintext and the response includes the unencrypted IV, an attacker who intercepts both the challenge (transmitted as plaintext) and response can easily recover the keystream used by XOR-ing the encrypted data frame with the plaintext challenge. The IV and keystream can be later reused for authenticating the attacker or injecting packets on the network.

However, even in open system mode, WEP turns out to be insecure. It has been shown that in a large set of RC4 keystreams, the first few bytes of the keystream are strongly nonrandom This property can be used to recover

information about the key by analyzing a high number of ciphertexts. To apply this attack to WEP, one needs to recover several thousand encrypted packets along with their IVs (at the time of this writing, the most recent attack can recover a WEP key with 50% probability using 40,000 data packets). If the network is not extremely busy, acquiring this many packets may take a very long time.

However, it is possible for an attacker to authenticate and associate to the access point (in open system mode, this does not require the WEP key), and then capture a single ARP packet (Section 5.2.3) from another client on the network. The attacker can then repeatedly transmit this packet to the access point, causing it to reply with a retransmission of this ARP packet along with a new IV. This attack is known as ***ARP reinjection***, and can allow an attacker to quickly capture enough IVs to recover the WEP key, at which point full access has been achieved and the attacker can perform additional attacks such as ARP cache poisoning.

On an idle network, capturing an ARP packet from a client may be difficult, especially if new connections are made infrequently. To speed up the process, an attacker can associate with the network and then send a deauthentication packet to the client, posing as the access point. The client will dutifully deauthenticate from the access point and reauthenticate, issuing a new ARP packet that can be captured by the attacker and retransmitted.

Another technique allows an attacker to decrypt WEP encrypted packets by exploiting the way in which the insecure CRC-32 checksum is handled. Recall that the CRC-32 checksum is appended to the data of the packet before encryption. Most access points silently drop packets with incorrect checksums. A technique known as the ***chop-chop*** attack uses this property of the access point to verify guesses of the packet contents. Essentially, the attacker truncates the data of the packet by one byte and corrects the CRC-32 checksum under the assumption that the dropped byte was a guessed byte x. This new packet is sent to the access point, and it will generate a response only if the guess x for the byte was correct. This guessing is repeated until the last byte is successfully guessed, at which point one byte of the keystream has been recovered. The entire process is then repeated until the entire keystream is recovered, at which point an ARP packet can be forged, encrypted with the keystream, and reinjected.

One final technique to compromise WEP security relies entirely on wireless clients, and requires absolutely no interaction with the targeted access point. The ***caffè latte*** attack, named for the fact that it could be used to attack clients in coffee shops with wireless access, relies on the fact that many operating systems feature wireless implementations that automatically connect to networks that have previously been connected to.

The attack takes advantage of this fact by listening to wireless traffic and identifying networks that a target client is attempting to connect to. The attacker then sets up a ***honeypot*** or ***soft access point***, a fake wireless access point with the same SSID as the AP the client is attempting to connect to, designed to lure in transmissions from the victim.

Recall that no stage of WEP authentication requires the access point to possess the WEP key. The client transmission is checked by the AP to confirm that the client possesses the key, but at no point does the client ever confirm that the AP has the key. As a result, the victim client in the caffè latte scenario associates and authenticates to the honeypot AP and sends a few ARP packets encrypted with the WEP key. However, in order to retrieve the WEP key, the attacker must have a high number of encrypted packets. To trick the client into sending these packets, the attacker takes the encrypted ARP requests received when the client connects to the honeypot AP and flips several predetermined bits. Specifically, the bits referring to the Sender MAC and Sender IP address are modified, and the CRC-32 checksum is recomputed using the chop-chop technique. This results in a valid encrypted ARP request with the client as the intended destination. The attacker then repeatedly sends this valid encrypted ARP request to the client, resulting in the client responding with enough encrypted ARP replies for the attacker to break the WEP key. After recovering the key, the attacker could modify the honeypot AP to actually use the key, allowing them to sniff and potentially modify traffic from the client, as in a man-in-the-middle scenario. The advantage of this attack over previous methods is that it does not require any interaction with the actual vulnerable wireless network. An attacker could use this technique to break an organization's WEP key without needing to be anywhere near the AP, as long as a client who had previously authenticated to that network was available.

6.5.3 Wi-Fi Protected Access (WPA)

Once the weaknesses in RC4 and WEP were published, IEEE quickly developed new standards that met more rigorous security requirements. The Wi-Fi Alliance then developed a protocol based on this standard known as ***Wi-Fi Protected Access*** (***WPA***). WPA is a more complex authentication scheme that relies on several stages of authentication. First, a shared secret key is derived for use in generating encryption keys and the client is authenticated to the access point. Next, this shared secret is used with an encryption algorithm to generate keystreams for encrypting wireless traffic. Finally, messages can be transmitted safely.

Authentication

WPA features two basic modes: ***PSK*** (***preshared key***) mode, also known as ***WPA Personal***, is designed for home and small office applications, while ***802.1x*** mode, also known as ***RADIUS*** or ***WPA Enterprise***, is ideal for larger networks and high-security applications. In 802.1x mode, a third-party authentication service is responsible for authenticating the client and generating key material. WPA allows integration with several choices for authentication mechanisms, each belonging to a framework known as the ***Extensible Authentication Protocol*** (***EAP***). The selected mechanism is invoked by the access point, and is used to negotiate session keys to be used by the client and access point during the next stage. 802.1x authentication protocols can make use of certificates and other elements from public-key cryptography to guarantee security. In PSK mode, a shared secret is established by manually entering a key into both the access point and the client.

Encrypting Traffic

The client and access point use the newly generated encryption keys to communicate over a secure channel. WPA has two possible protocols for encrypting traffic. The ***Temporal Key Integrity Protocol*** (***TKIP***) makes use of RC4 and was designed to provide increased security over WEP while remaining compatible with legacy hardware. Newer hardware supports a standard called ***WPA2***, which features a stream cipher based on AES and a cryptographically secure MAC based on AES for message integrity.

TKIP attempts to address the cryptographic weaknesses of WEP's RC4 implementation. WEP is especially weak because it simply concatenates the IV with the encryption key to generate the RC4 seed. TKIP remedies this by increasing the IV length to 48 bits and by incorporating a key-mixing algorithm that combines the key with the IV in a more sophisticated way before using it as an RC4 seed to generate a keystream. In addition, TKIP replaces the CRC-32 checksum with a 64-bit ***message integrity code*** (***MIC***) computed with the ***MICHAEL*** algorithm, which was designed to serve as a mesage authentication code (MAC) (Section 1.3.4) computed from the message and a secret 64-bit key. The MICHAEL algorithm has been shown to be cryptographically insecure. However, attacks against MICHAEL are much more difficult to accomplish than attacks against CRC-32.

When an access point receives a packet with a nonmatching MIC, countermeasures such as alerting a network administrator or regenerating the PTK may be invoked. Finally, TKIP implements a sequence counter in the

packets to prevent replay attacks. If a packet is received out of order, it is simply dropped.

TKIP provides many improvements over standard WEP encryption, but its use is now deprecated in favor of the newer WPA2 protocol. While TKIP relies on the RC4 cipher and MICHAEL algorithm (both efficient but cryptographically weak), WPA2 instead uses the strong AES cipher for protecting both integrity and confidentiality. The AES cipher has not been broken as of the time of this writing.

Attacks on WPA

Currently, WPA in 802.1x mode is considered secure. However, PSK mode may be vulnerable to password cracking if a weak password is used and an attacker can capture the packets of the initial four-way handshake that authenticates the target to the access point. Once this handshake is captured, the attacker can launch a dictionary attack against the encrypted messages. However, this attack is made more complicated by the mechanism used to convert the user-supplied key, which may be as simple as a dictionary word, into the necessary 256-bit string. The key can be provided directly as a string of 64 hexadecimal digits (which would make any dictionary attack infeasible), or may be provided as a passphrase of 8 to 63 ASCII characters.

In the event that the passphrase is entered using ASCII, the key is calculated by using the SSID of the access point as a salt for a key derivation function known as ***PBKDF2***, which uses 4,096 iterations of the ***HMAC-SHA1*** hash as a salt is designed to prevent dictionary attacks relying on extensive precomputation. However, researchers have published tables of precomputed keys corresponding to the most popular SSIDs. In addition, if the password used is a simple dictionary word, an attacker could recover the password using a dictionary attack without the use of any precomputation. This is not considered a weakness in WPA itself, but rather serves as a reminder that strong passwords should be used to prevent dictionary attacks.

More recently, researchers discovered a vulnerability in TKIP that allows an attacker to recover the keystream used for a single packet (as opposed to the key used to seed that keystream), allowing that attacker to transmit 7–15 arbitrary packets on that network. The attack stems from the fact that for compatibility purposes, TKIP continues to utilize the insecure CRC-32 checksum mechanism in addition to the improved MICHAEL algorithm.

Just as with the chop-chop attack on WEP, an attacker uses the fact that access points may drop packets that do not have valid CRC-32 checksums to his advantage. The attacker captures an ARP packet, which is easily

identified by its length. In fact, the contents of ARP requests are mostly known by the attacker ahead of time, with the exception of the last bytes of the source and destination IP addresses, the 8-byte MICHAEL algorithm, and the 4-byte CRC-32 checksum. Using a variation on the chop-chop method, the attacker guesses values for these unknown bytes, using the access point to verify each guess.

However, TKIP has an additional defense mechanism that issues a warning and regenerates encryption keys when two messages with the correct CRC-32 but incorrect MICHAEL checksum are received within the same minute. To circumvent this, the attacker can simply wait 1 minute between each guessed value. Once the packet has been decrypted, the attacker has recovered both the keystream and the MICHAEL key used to generate the packet's checksum. Using this information, the attacker can craft and transmit 7–15 arbitrary packets to the network. This attack can be prevented by configuring TKIP to reissue keys at short intervals, or by switching to the more secure WPA2 protocol, that no longer uses CRC-32.

6.6 Exercises

For help with exercises, please visit **securitybook.net**.

Reinforcement

R-6.1 Describe the main purpose of DNS.

R-6.2 Suppose the transaction ID for DNS queries can take values from 1 to 65,536 and is randomly chosen for each DNS request. If an attacker sends 1,024 false replies per request, how many requests should he trigger to compromise the DNS cache of the victim with probability 99%?

R-6.3 Why are pharming and phishing attacks often used in concert with each other?

R-6.4 Give three different techniques that an attacker can use to make a victim send DNS requests to domains chosen by the attacker.

R-6.5 Explain the difference between the subdomain DNS cache poisoning attack and the traditional version of this attack.

R-6.6 Compare and contrast the way a regular DNS request is answered and the way it would be answered and authenticated in DNSSEC.

R-6.7 Explain how a stateless firewall would block all incoming and outgoing HTTP requests.

R-6.8 How can SSH be used to bypass firewall policy? What can a network administrator do to prevent this circumvention?

R-6.9 Describe a firewall rule that can prevent IP spoofing on outgoing packets from its internal network.

R-6.10 What is the difference between a misfeasor and clandestine user?

R-6.11 Explain how a port scan might be a preliminary indication that another attack is on its way.

R-6.12 Which is worse for an intrusion detection system, false positives or false negatives? Why?

R-6.13 Give examples of IDS audit records for each of the following actions:

(a) A user, Alice, reading a file, foo.txt, owned by Bob, of size 100 MB, on December 18, 2010

(b) A client, 129.34.90.101, initiating a TCP session with a server, 45.230.122.118, using 0.01 CPU seconds, on January 16, 2009

(c) A user, Charlie, logging out from his computer, using 0.02 CPU seconds, on March 15, 2010

R-6.14 What are the main differences between WEP and WPA? What are the different possible modes under the WPA standard?

R-6.15 Explain why deep packet inspection cannot be performed on protocols such as SSL and SSH.

R-6.16 Explain how IP broadcast messages can be used to perform a smurf DOS attack.

R-6.17 How does a honeypot fit in with the security provided by a firewall and intrusion detection system?

Creativity

C-6.1 Suppose DNS IDs were extended from 16 bits to 32 bits. Based on a birthday paradox analysis, how many DNS requests and equal number of fake responses would an attacker need to make in order to get a 50% chance of succeeding in a DNS cache poisoning attack?

C-6.2 Explain why a large value for the TTL (time-to-live) of replies to DNS queries does not prevent a DNS cache poisoning attack.

C-6.3 Suppose Alice sends packets to Bob using TCP over IPsec. If the TCP acknowledgment from Bob is lost, then the TCP sender at Alice's side will assume the corresponding data packet was lost, and thus retransmit the packet. Will the retransmitted TCP packet be regarded as a replay packet by IPsec at Bob's side and be discarded? Explain your answer.

C-6.4 An alternative type of port scan is the ***ACK scan***. An ACK scan does not provide information about whether a target machine's ports are open or closed, but rather whether or not access to those ports is being blocked by a firewall. Although most firewalls block SYN packets from unknown sources, many allow ACK packets through. To perform an ACK scan, the party performing the scan sends an ACK packet to each port on the target machine. If there is no response or an ICMP "destination unreachable" packet is received as a response, then the port is blocked by a firewall. If the scanned port replies with a RST packet (the default response when an unsolicited ACK packet is received), then the ACK packet reached its intended host, so the target port is not being filtered by a firewall. Note, however, that the port itself may be open or closed: ACK scans help map out a firewall's rulesets, but more information is needed to determine the state of the target machine's ports. Describe a set of rules that could be used by an intrusion detection system to detect an ACK scan.

C-6.5 During a ***FIN scan***, a FIN packet is sent to each port of the target. If there is no response, then the port is open, but if a RST packet is sent in response, the port is closed. The success of this type of scan depends on the target operating systems—many OSs, including Windows, have changed the default behavior of their TCP/IP stacks to prevent this type of scan. How? Also, how could an intrusion detection system be configured to detect a FIN scan?

C-6.6 Explain how it would give a potential intruder an additional advantage if he can spend a week stealthily watching the behaviors of the users on the computer he plans to attack.

C-6.7 Describe the types of rules that would be needed for a rule-based intrusion detection system to detect a DNS cache poisoning attack.

C-6.8 Describe the types of rules that would be needed for a rule-based intrusion detection system to detect an ARP spoofing attack.

C-6.9 Describe the types of rules that would be needed for a rule-based intrusion detection system to detect a ping flood attack.

C-6.10 Describe the types of rules that would be needed for a rule-based intrusion detection system to detect a smurf attack.

C-6.11 Describe the types of rules that would be needed for a rule-based intrusion detection system to detect a SYN flood attack.

C-6.12 The ***coupon collector*** problem characterizes the expected number of days that it takes to get n coupons if one receives one of these coupons at random every day in the mail. This number is approximately $n \ln n$. Use this fact to compare the number of TCP connections that are initiated in a sequential port scan, going from port 1 to 65535, directed at some host, to the expected number that are requested in a random port scan, which requests a random port each time (uniformly and independently) until it has probed all of the ports.

C-6.13 Describe a modification to the random port scan, as described in the previous exercise, so that it still uses a randomly generated sequence of port numbers but will now have exactly the same number of attempted TCP connections as a sequential port scan.

Projects

P-6.1 Keep a diary that chronicles how you use your computer for an entire week. Try to include all the key elements that are included in an intrusion detection event log, including which files you read

and write, which programs you run, and which web sites you visit. (Your browser probably keeps a history of this last set of events itself.) Write a term paper that discusses, at a high level, the types of rules and statistics that could be used to build an intrusion detection system for your computer that could tell if someone else was using it besides you. Include a discussion of how easy or difficult it is to predict normal and anomalous behavior for your computer based on your usage patterns for this week.

P-6.2 Write a term paper that discusses the risks of pharming and phishing with respect to identity theft, including spam emails claiming to come from well-known companies and financial institutions. Include in your paper a discussion of some of the current techniques being deployed to reduce pharming and phishing, including how effective they are.

P-6.3 On an authorized virtual machine network, define three virtual machines, DNS Server, Victim, and Attacker, which could in fact all really be on the same host computer. On DNS Server, install the DNS server software (such as `bind`), and configure the DNS server to respond to the queries for the `example.com` domain. (It should be noted that the `example.com` domain name is reserved for use in documentation and is not owned by anybody.) Configure Victim so it uses DNS Server as its default DNS server. On Attacker, install packet sniffing and spoofing tools, such as `Wireshark` and `Netwox`. Let Attacker and Victim be on the same LAN, so Attacker can sniff Victim's DNS query packets, and have Attacker launch DNS attacks on Victim. Once this succeeds, let Attacker and Victim be on two different networks, so Attacker cannot see the Victim's DNS query packets. Have Attacker launch DNS attacks on Victim in this more difficult setting.

P-6.4 On a virtual machine, install the Linux operating system. Implement a simple, stateless, and personal firewall for this machine. The firewall inspects each packet from the outside, and filters out the packets with the IP/TCP/UDP headers that match the predefined firewall rules. The Linux built-in `Netfilter` mechanism can be used to implement the firewall.

P-6.5 On a virtual machine, install the Linux operating system. Linux has a tool called `iptables`, which is essentially a firewall built upon the `Netfilter` mechanism. Develop a list of firewall rules that need to be enforced on the machine, and configure `iptables` to enforce these rules.

P-6.6 Design and implement a program to do DNS lookups, and simulate a DNS poisoning attack in this system.

P-6.7 Write a web crawler or collect enough spam emails for yourself and friends in order to find five phishing web sites. Compare the content of these pages with their authentic counterparts, both in terms of HTML source and the look and feel of the pages as displayed in the browser.

P-6.8 Write a client/server program that tunnels data using a nonconventional protocol. For example, create a messaging program that sends data in the payload of ICMP packets.

Chapter Notes

Several of the protocols mentioned in this chapter are documented with RFCs:

- RFC 1035 - DNS
- RFC 2460 - IPv6 and IPSec
- RFC 4251 - SSH

For more details on the topics covered in this chapter, see the book by Cheswick, Bellovin and Rubin [16] and the previously cited books by Comer [18], Tanenbaum [100], Kaufman, Perlman, and Speciner [46], and Stallings [96]. Lioy et al. present a survey of DNS security [57]. Dan Kaminsky discovered the subdomain resolution attack for cache poisoning and collaborated with major providers of DNS software on the development of patches before a making public announcement of the vulnerability in 2008. Keromytis et al. discuss implementing IPSec [48]. Martin Roesch, lead developer of the Snort intrusion detection system, describes its goals and architecture [84]. Niels Provos presents a framework for virtual honeypots [78]. Attacks on WEP are given by Borisov, Goldberg and Wagner [12] and by Stubblefield, Ioannidis and Rubin [98].