Network Security

December 14, 2016

1 Ethernet

The Internet is implemented in a hierarchical set of layers of abstraction in which protocols use functionality of the lower layers to send messages over a network, culminating in the highest layer, known as the application layer, which provides foundational protocols such as FTP and HTTP which are used by web servers, browsers, and other end users. In this section, we'll focus on the lowest levels of the stack: the physical and link layers. Let's begin with a high level overview of what happens when a computer connects to the Internet over WiFi.

1. Joining a network: First, the computer scans for routers on open networks (i.e. routers accepting connections). It does so by broadcasting packets asking if a particular wireless network exists. Routers are also continuously broadcasting that they are on a network accepting connections. If a router on the same wireless network as the one the computer is searching for exists, it will recieve the computer's message (and the computer will recieve) the router's, and the computer will join the network.

2. Connection configuration: The computer now broadcasts to the local network on which it's connected asking what configuration it needs to use. The configuration details cover information such as the computer's IP address, it's gateway, the locations of DNS servers, etc. DHCP servers on the local network reply back with the computer's configuration information, including their assigned IP address and other relevant data.

3. DNS requests: This step and the following detail what occur when the computer attempts to navigate to a website. First, the URL needs to be parsed into an IP address and some other relevant metadata. This is done using DNS, which uses the UDP internet protocol. The computer sends its DNS request, which asks what the IP address of the URL is, to a DNS server. Specifically, the request is sent to a recursive resolver, which iteratively narrows down which name server contains the mapping from the URL to IP address. When the IP address has been resolved, it's send back to the computer.

4. TCP: Now that the computer has the IP address of its intended host, it needs to establish a TCP connection. It does so by initiating the TCP handshake with a SYN request, which the server responds to which a SYN/ACK, followed by a final SYN reply by the computer.

5. TLS: Now that a TCP connection has been established, the computer may communicate with the server. When using secure protocols such as HTTPS, the next step is to negotiate an encrypted TLS session that runs on top of the TCP connection. The computer initiates the TLS handshake, the server replies with its public key and a signed certificate proving that it is who it says it is. The handshake continues until a symmetric key exchange is confirmed, after which the computer and server may communicate securely.

6. HTTP: The computer can now send (encrypted) HTTP requests asking the server for the data specified in the URL, which the server then replies to in an HTTP response.

One important property of the Internet protocol, which runs right beneath TCP, is that it's packet switched, which means all communicated data is broken into variable length (up to a maximum) packets of data. The data is disassembled before being sent along the network reassembled when they arrive at their destination; in between, the network simply sends the packets on their way to the destination, not accounting for order. Moreover, packets include a checksum which is used to check for random errors or data corruption; in such cases, the packets are dropped entirely. Thus, packets are either received correctly or not at all. It's up to higher level protocols, such as TCP, to maintain reliability by resending packets that don't make it.

Packet sniffing: One security flaw is that when the computer broadcasts packets meant for the router, or even connection requests or information in the early stages of joining a wireless network, the packets are broadcast everywhere at once, to all listening devices (this is unavoidable since the packets are transmitted through omnidirectional radio waves, but even on ethernet the behavior is for all packets on a local network to be sent to a local hub accessible by every other node on the local network). Although network cards are by defualt instructed to ignore packets from unestablished connections, they can be set to *promiscuous mode*, allowing them to eavesdrop on any packets they pick up.

Moreover, when a computer initially broadcasts requests to join a connection, it will simply accept the first reply it hears. Thus, a malicious attacker would impersonate the router and get the computer to route all its traffic through the attacker. Such attackers are known as *rogue access points.* This speaks to a more general concern - packet injection. Low level network layers don't enforce integrity or authenticity, allowing attackers to impersonate other devices and inject spoofed packets into a connection. In many cases, when a computer broadcasts a request, it will accept the first reply it hears, creating a race condition between an attacker and the genuine response. This can be defended against by encrypted all traffic. Moreover, injected packets can be detected, since they don't stop the genuine packets from also arriving at their destination, but it's often too late by that time.
To avoid packet injection, secure WiFi protocols such as WPA2 establish a pre-shared key, or PSK, which allows them to create a symmetric shared key, known as the PTK (pairwise transient key). Injected packets won't have the correct PTK whereas genuine ones will, allowing the computer to filter out injected packets. Secure LANs avoid the process entirely by using *network switches*, which are devices which replace the network hub (which broadcasts all data to every node) and remember mappings between MAC addresses and connections, forwarding the relevant data to only the right MAC addresses. Thus, if all communications are sent through a network switch, only the right recipients will receive their data.

2 DNS

The *domain name system,* or DNS, is the protocol used to resolve URLs, which are abstract locators of hosts on the Internet, to the actual IP addresses of the corresponding servers. Clients make DNS requests, asking for the IP address corresponding to a URL, to DNS *name servers.* These requests are sent using a lower level protocol known as the *user datagram protocol*, or UDP. UDP is on the same abstraction layer as IP, the Internet Protocol (which runs under TCP), and is fast but unreliable. UDP packets may or may not be delivered, and are unordered. In contrast to TCP, which takes various measures to ensure that data is guaranteed to arrive and isn't corrupted during transit, UDP is similar to IP in that it requires no handshake or other overhead to establish and consists only of a rapid fire stream of packets sent to the destination, out of order and without measure to ensure against corruption or packet loss.

There are two types of DNS servers - recursive DNS servers and authoritative servers. When a client requests the IP address associated with a URL, it sends a DNS request to a recursive DNS server, also known as a recursive resolver. IP addresses are structured hierarchically in a tree structure to make their organization more efficient. At the root is the root server, represented by a `.`, so all URLs have an implicit `.` at the end. There are only about a dozen root name servers in the world. Beneath the root server are *top level domains*, or TLDs, such as `.com`, `.gov`, etc., below which are more general servers, such as `google.com`, which preside over company specific servers. Authoritative DNS servers are the servers which actually present the relevant mapping from URL to IP address; the purpose of recursive DNS servers is, given an IP address representing a level in the hierarchy, to iteratively query authoritative name server for the IP address of the authoritative DNS server the level (in the hierarchy) below, until it reaches the end of the IP address and finds an authoritative name server with the exact IP address the client wanted in the first place, which is then returned to the client in a DNS response. Recursive resolvers, authoritative DNS servers, and clients (e.g. browsers) employ DNS caching, saving the IP address of URLs the first time they're resolved. Recursive lookup is problematic from a security standpoint because if any of the name servers in the tree are malicious, they can spoof DNS replies that send the victim to a malicious website. Malicious DNS servers are rare, but we still need to worry about attackers eavesdropping on DNS traffic, or spoofing their own and injecting DNS replies.

Malicious DNS servers: The primary security concern here is that if any of the DNS servers queried by the recursive resolver are malicious, they can poison our cache by lying about the true IP address of the URL, since there's no way of verifying if the IP address is the right one without already having it. To address the first problem, we enforce a policy known as *bailiwick checking.* DNS responses to a recursive resolver have three sections - an answer section, an authority section, and an additional section. The answer section contains the name server below it to query next, which is what the recursive resolver asked for. The other two sections contain the IP addresses of other relevant name servers, such as a list of authoritative name servers (which provide IP address lookups, butonly to original sources and not recursive resolvers). Malicious DNS servers could inject any arbitrary mapping between a URL and IP address in the two auxiliary sections, along with the correct response in the answer section, thereby silently poisoning the DNS cache of the victim for a different URL. This is prevented with bailiwick checking, where recursive resolvers simply ignore any mappings in a DNS reply which try to give lookups for an IP address that aren't in the same domain that the original DNS request asked for.

On-path eavesdroppers: Attackers that are **on-path**, meaning they are privy to all the DNS packets we send to a recursive resolver, they can create a race condition and spoof DNS responses, and since by default the first DNS response to a DNS query is accepted, they might succeed in poisoning our DNS cache.

Off-path eavesdroppers: DNS packets carry a *transaction identifier* for security purposes. The transaction identifier simply a 16-bit number sent with DNS queries; we expect DNS responses to have the same transaction identifier, and if it doesn't, we ignore the response. If the transaction identifier is randomly chosen, this provides some security against off-path eavesdroppers since they won't know the transaction identifier without seeing the initial DNS request, and so are very unlikely to succeed in spoofing a DNS response with the correct identifier. Of course, this doesn't protect against on-path attackers, who would see the transaction identifier. However, it's possible even for off-path attackers to circumvent this, using a technique known as *blind spoofing*.

The way blind spoofing works is that if an attacker can predict when we'll make a DNS request, they can flood our computer with spoofed DNS responses with various transaction identifiers, and hope that one of them by random chance will be the right one. Situations where attackers can predict when a victim makes a DNS query aren't uncommon; in many cases, the attacker can, at will, get a victim to make a DNS query (e.g. if the victim visits a website under the attacker's control, the webpage could reply with HTML elements that force the browser to perform HTTP requests and hence DNS lookups). If transaction identifiers aren't randomly iterated (previously, they used to be simply incremented between DNS requests), the attacker could force us to make a decoy DNS request to a server he controls, note the transaction ID, and compute the next one when they spoof DNS responses. Even when transaction IDs are randomly selected, however, implementation bugs can enable attackers to blindly spoof packets. Because DNS runs on UDP, the victim's computer is constantly listening for UDP packets, expecting a DNS reply, without any connection establishment or security measures, allowing an attacker to rapid-fire thousands of spoofed DNS packets in quick succession. Since recursive lookups take some time, if the attacker can narrow the search space of transaction IDs, he might be able to cover a significant fraction of it before the recursive resolver replies with the legitimate response, giving the attacker a reasonable probability of success.

- Kaminski attack: The first real world instance of blind spoofing of the above form was discovered by a researcher named Dan Kaminski. He noticed that although the chance of finding the correct transaction ID for a given spoofed DNS response is small - 1 in 65,636 - it's possible to try multiple times. This is because rather than trying to poison the cache entry of, say, `google.com`, we can poison the cache entry of `a1.google.com` and list `google.com` as an authority. This will be accepted by the recursive resolver since, after all, `a1.google.com` is in-bailiwick. If the attacker can get 1,000 packets in before the legitimate reply, he has about a 1 in 65 chance of success. If he fails, however, he can try again with `a2.google.com`, and so on until he succeeds, with the probability of success growing higher and higher with each attempt. In practice, getting a spoofed DNS response with the correct transaction ID takes about ten seconds with this technique.

The Kaminski attack and related blind spoofing attacks, while still theoretically possible, have been mitigated by simply increasing the search space of the transaction IDs to 32 bits (they now include a random UDP source port to send to as well, rather than a fixed one).

2.1 DNSSEC

DNS is designed with robustness and scalability and not security in mind, so in an attempt to address the numerous security flaws in DNS, the Internet Engineering Task Force (IEFT) came out with a set of specifications, known as the Domain Name System Security Extensions (DNSSEC). DNSSEC is meant to be a set of extensions to DNS which provide DNS clients with origin authentication of DNS data and data integrity, though not availability or confidentiality. Moreover, TCP and higher level protocols, which support cryptographic protocols to run on top of them, weren't used for DNS packet transmission for latency concerns (such a structure would be several times slower than the existing UDP base). Loosely speaking, DNSSEC is a hierarchical and distributed trust system designed to validate the mappings of names to values. One way data integrity is ensured is that DNS domain servers in DNSSEC zones are required to sign the information returned by any name server in its domain (e.g. root server signs the key for `.com`, which signs the key for `google.com`, and so on). The recursive resolver validates every response it gets from an authoritative DNS server by validating the digital signature. Clients have the digital signature key for root servers hard-wired (this is possible since there are only a few root servers, and they don't change), and can then recursively leverage each authoritative DNS server's digital key to validate the key of the server in the hierarchical level below it.
This process can be very fast, especially since it signature validation can be done while the recursive resolver is waiting for the DNS reply of the next authoritative server.

3 TCP/IP

Like UDP, IP packets don't use any form of connection establishment or security, and are fast but unreliable. Packets may be corrupted or dropped entirely during transit. Moreover, IP packets carry their source and destination IP address on them, and nodes in the middle of the network will have no way of verifying the veracity of the addresses. Thus, if the node directly connected to the attacker's computer doesn't verify IP packets, the attacker can simply spoof IP packets with a fake source IP address and impersonate another server. This also enables reflection attacks, where the attacker can get a server to send packets to a victim's IP address, by spoofing the source IP of IP packets with predictable responses to the victim's IP; this is useful in denial of service (DOS) attacks.

TCP (transmission control protocol), which runs on top of IP, is more reliable, and sends packets in order and with checksums to verify data corruption (these aren't message authentication codes, however, and don't protect against intentional adversarial corruption). TCP connections, once established, are characterized by the source and destination ports, as well as random sequence numbers exchanged between the participants. If an attacker can find the ports and sequence numbers, he can inject TCP packets into the connection. Thus, being able to eavesdrop on TCP packets is equivalent to being able to inject TCP packets. Blind spoofing attacks are also possible; since sequence numbers are based on pseudo-random number generators, if an attacker can find out the initial seed, they can compute the sequence numbers. Inferring the seed isn't impossible, depending on the protocol used, and the attacker can first set up a non-spoofed TCP connection with a victim to try and test how the computer is generating seeds (the default is to use the system clock, which is a problem since an attacker could potentially predict when a victim will try to establish a TCP connection and guess the seed). This isn't too big a problem today, as the randomness in seeds is quite good, but on-path eavesdroppers being able to inject packets remains a major issue.

Distributed denial of service: Otherwise known as DDOS attacks, such attacks consist of flooding a victim's system with data to make it unusable to its indented users. If the attacker only has a few systems sending the flood of packets, the victim can simply ban their IPs. However, in a DDOS attack the attacker is assumed to have access to a large distributed network of bots, making defense very difficult. The attacker can also amplify his computational power using reflection attacks, since in many cases packet responses are much larger than the initial packets sent (e.g. DNS packet responses), by spoofing the source IP to be the victim's IP address. There are various techniques attackers may employ within the DNS framework to inflate the size of the response packets, making the DDOS attack more potent. Attackers could also try to overwhelm the victim's memory capacity. One way to do this is known as a *SYN flood*, where an attackers sends an onslaught of many TCP SYN packets to the victim; each packet is stored in memory and burdens the system. Such attacks can be resisted with SYN cookies - rather than dropping connections once the SYN queue is full, it encodes the state of the connection (souce and destination IP addresses, etc.) and replies with a SYN/ACK and then deleting the SYN packet, effectively behaving as if the SYN queue had been enlarged. A connection is only established if the attacker replies with an ACK packet that contains the correct state.

4 TLS

TLS (transport layer security) is a cryptographic protocol which runs on top of TCP, and can be used to provide confidentiality, integrity, and authenticity. Notice that this is the first instance of any cryptographic protocol implementation in the stack so far; this is why all layers below the transport layer are vulnerable to so many attacks. The idea is simple - once a TCP connection is established, the server sends over a list of supported cryptographic protocols that it supports. the client then chooses a protocol to use, and sends it over to the server. Next, the client chooses random symmetric keys for encryption and authentication. The keys are encrypted with public key encryption, and sent to the server. This requires the server to send its public key to the client, as well as a digital signature (which the client validates) to verify that the public key really does belong to who the server claims it belongs to. All subsequent communication involves symmetric encryption and MACs.

This protects against packet spoofing, eavesdropping, and injection. The only security concern remaining is that a full man-in-the-middle (MITM) attacker could inject RST (reset) packets, terminating the TCP connection. Thus, even full MITM attackers are limited to denial of service attacks. Moreover, if an attacker were to somehow gain access to the private key associated with a certificate, the attacker would fully impersonate website holding that certificate would be.