Paxson
Spring 2011

CS 161
Computer Security

Discussion 4

February 16, 2011

Question 1 ***Firewall Misconfiguration*** **(8 min)**

Suppose you had a rule set that looked like this:

1. drop tcp 10.1.1.0/25:* *:*
2. allow udp *:* 192.168.1.0/24:*
3. drop tcp 10.1.1.128/25:* *:*
4. drop udp 172.16.1.0/24:* 192.168.1.0/24:*
5. allow tcp 10.1.1.0/24:* *:*
6. drop udp 10.1.1.0/24:* 192.168.0.0/16:*
7. allow udp 172.16.1.0/24:* *:*

Which rules contradict one another? Can you find more than one example?

Hint: Look for when all the packets one rule intends to deny (accept) gets accepted (denied) by a preceding rule?

Solution: Rules 2 and 4 contradict each other. All packets to 192.168.1.0/24 are accepted by rule 2, but rule 4 denies the subset of packets from 172.16.1.0/24 to 192.168.1.0/24. This is called *shadowing*.

Rule 5 is shadowed by the combination of rules 1 and 3. Rule 1 denies any IP that matches the first 24 bits 10.1.1, followed by a single 0 bit. Rule 3 denies any IP that matches the first 24 bits 10.1.1, followed by a 1 bit. Combined, this is equivalent to denying any IP that matches 10.1.1, which is what rule 5 states.

Question 2 ***Denial-of-Service (DoS)*** **(7 min)**

In 2007, a large broadband provider in the United States was attacking end users' BitTorrent connections with a layer-4 Denial-of-Service (DoS) attack. For each BitTorrent connection, the provider injects a single packet into the network, which tears down the connection.

(a) How might this have worked?

(b) Does TCP defend against this attack? Explain how it does or why it does not.

(c) How could a different attacker (not the ISP) launch this attack? Assume that the attacker is another home broadband user.

Solution:

(a) The ISP can send a single RST packet to tear down a TCP connection, as specified by the protocol.

(b) TCP cannot defend against this attack because the ISP is in the path between the two connection endpoints. The ISP only needs to look at the TCP sequence number and inject a packet with the right sequence number.

The ISP does not need to resort to blind spoofing, i.e., guess the correct sequence number. This is true for any eavesdropping adversary and in particular for a man-in-the-middle (MITM) attack.

(c) A different attacker who does not sit in the path between the two connection endpoints would have to blind spoof a TCP sequence number (i.e., guess the TCP right sequence number) in order to inject a RST packet that would be accepted. This is much harder.

Question 3 ***Circumventing Network Policy Controls*** **(5 min)**

You are stuck at the Chicago O'Hare airport for at least 7 more hours, and the airport does not have free WiFi. How boring! Since you are unwilling to pay the offensive fees the hotspot provider demands to access the Internet, you start to explore what you can do with the limited connectivity. Quickly you find out that all connections to Internet hosts are blocked. You also discover that the DNS server of the hotspot answers queries for any hostname in the Internet.

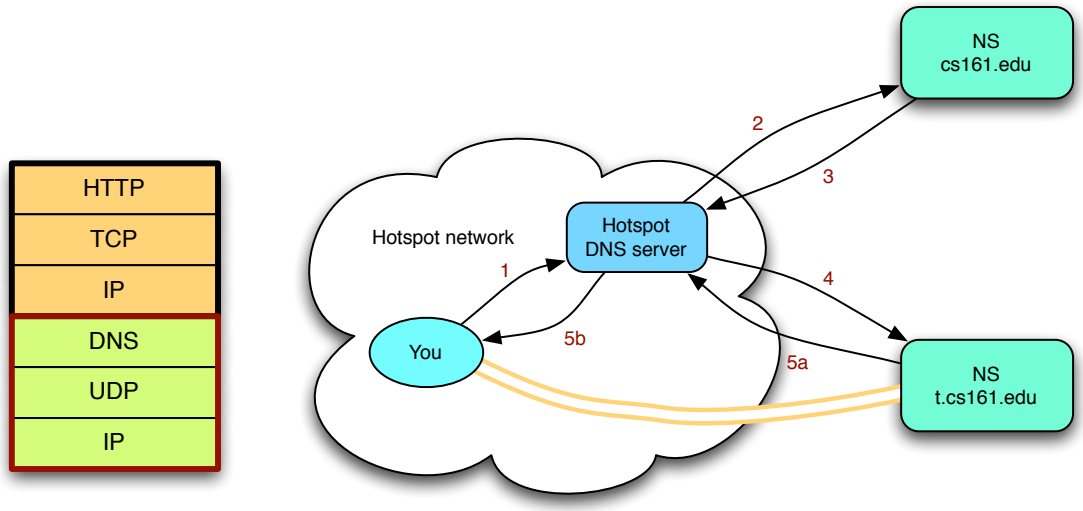
As an exceptionally studious and forward-looking CS 161 student, you were prepared for this scenario of Internet deprivation and set up your own DNS infrastructure prior to leaving on your trip. How is it possible to get free WiFi access in this scenario? What needed to happen prior to the trip?

Solution: In this scenario, you can tunnel Internet traffic through DNS. The key idea is to encapsulate your traffic into DNS queries that you send to a DNS server in the Internet under your control. This server decapsulates the DNS queries and relays the data to its intended destination, stuffs the response traffic into a DNS reply and ships it back to you.

For this to work, you must have setup a DNS server and a tunnel domain prior to leaving on the trip. Assume you own the domain `cs161.edu` and you would like to use `t.cs161.edu` as the tunnel subdomain.[1] Further, assume that the IP address of

`t.cs161.edu` is 6.6.6.6 and you have deployed a custom DNS server at that address listening on port 53 for incoming requests.

The following graphic illustrates the necessary steps to establish a DNS tunnel. On the left hand side you see the network stack, where the bottom three layers correspond to the regular DNS communication in the hotspot network, depicted by the black arrows in the Figure on the right. The top three layers represent the *overlay network* that you create *inside* the DNS communication. In this case, assume that you would like to establish a HTTP connection to `128.32.244.172` and have already created a HTTP request with appropriate TCP/IP header. For simplicity, let us refer to this bundled data as `encoded-request`.



1. To get the data to the server, you ask the local hotspot resolver to lookup `encoded-request.t.cs161.edu`.
2. Assuming the hotspot resolver already knows the nameserver for `cs161.edu`, the resolver now needs to find out who is responsible for `t.cs161.edu`.
3. The authorative nameserver for `cs161.edu` tells the hotspot resolver that `t.cs161.edu` is managed by 6.6.6.6.
4. Next, the hotspot resolver sends the query for `encoded-request.t.cs161.edu` to 6.6.6.6. Your custom nameserver at this address knows how to decode the incoming data, forward the HTTP request to `128.32.244.172` and encode the HTTP reply plus the TCP/IP headers.

5a. The encoded response is sent back as a TXT record, another form of DNS record we have not yet encountered. This record allows you to associate arbitrary text with a DNS name.

5b. After having received the encoded HTTP response as TXT record, the hotspot resolver forwards it you. At this point, you can decode the data in the TXT record and rebuild the overlay TCP/IP stack to extract the HTTP reply.

Note that steps (2) and (3) only occur once because the hotspot resolver will cache the nameserver for `t.cs161.edu`.

If you would like to read more about DNS tunneling, you can find a good introduction at `www.dnstunnel.de`. Moreover, the software *iodine* [1] is a concrete example of a DNS tunnel implementation that ships both a client to be used on your laptop and a custom DNS server daemon.

References

[1] Kyro. iodine. `http://code.kryo.se/iodine`.

[1] One often uses a separate subdomain for tunneling to be able to use the main domain also for other purposes. Another reason is that some (web-based) domain management tools only allow you to set NS entries for subdomains but not for the top-level domain itself.