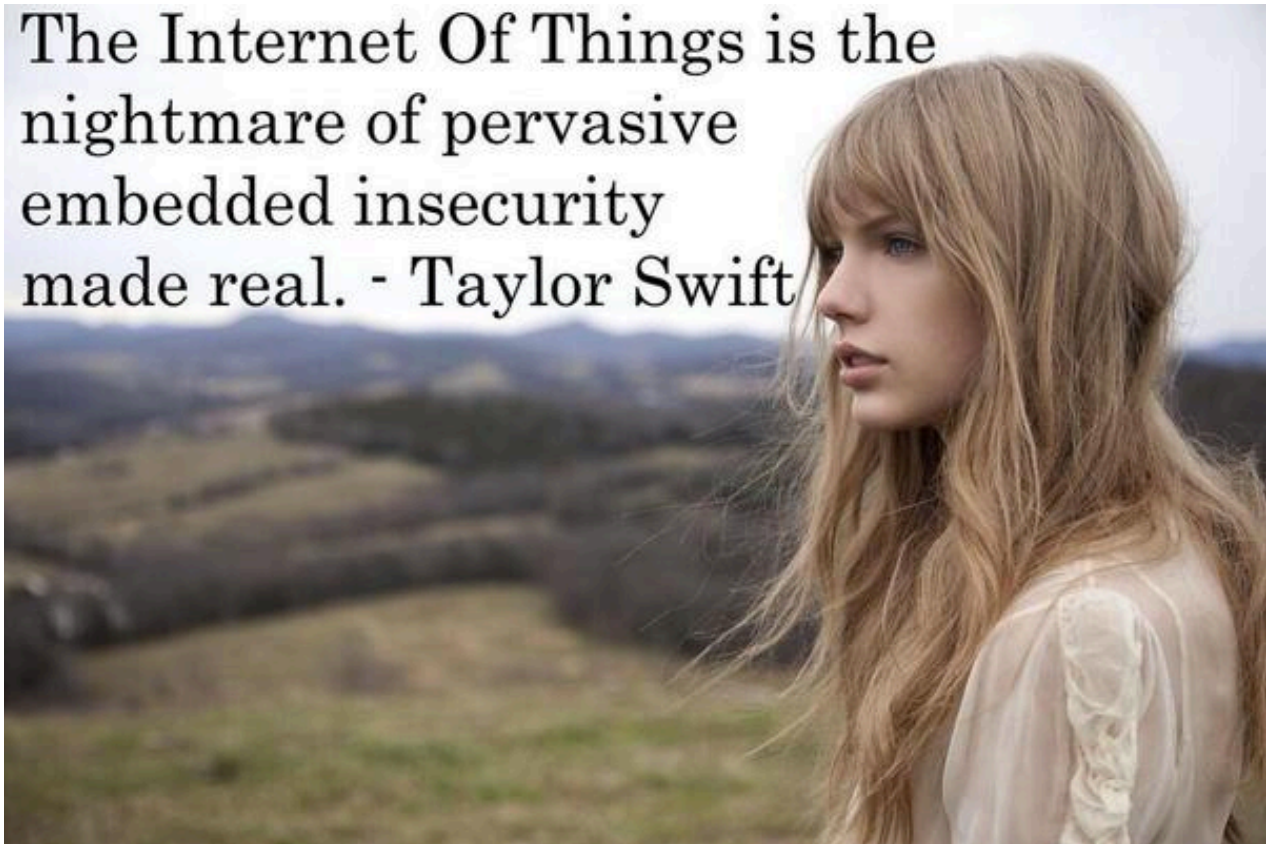
The Net Part 2



Writing IP addresses

- IPv4 -> 32b
  - `aa.bb.cc.dd`
    - Decimal values from 0-255, e.g. `128.32.131.12`
- IPv6 -> 128b
  - `aaaa:bbbb:cccc:dddd:eeee:ffff:gggg:hhhh`
    - Hexadecimal values (can drop leading 0), e.g. `2607:f140:2000:4001:187f:86cc:3dfc:b9c8`
    - A long run of 0s can be replaced with ::
- Subnets (/8, /16, /24...)
  - `128.32/16`
    - All IPv4 between `128.32.0.0` and `128.32.255.255`
  - `2607:f140:2000:4001/64`
    - All IPv6 addresses with the same upper 64 bits
  - Also subnet masks: 255.255.255.0 is the /24 subnet mask: Specify which bits are in the local network address

Special IP addresses & Networks

- Localhost: `127.0.0/24`
- Broadcast: `255.255.255.255`
  - Send to all in the local network
  - Also for subnet, can specify all bits as 1 (e.g. for `128.32/16`, `128.32.255.255`) to broadcast to that network, but generally ignored these days
- Private: `10/8`, `172.16/12` (ends up being .16-.32), `192.168/16`
  - Not routed on the Internet, can use for internal purposes
  - Commonly used for NAT (more later)
- IPv6 Multicast: `ff00:/8`
  - In particular `ff00::1` -> all machines on local network
- Thus stay at `127.0.0.1`
  and wear a `255.255.255.0`

Physical/Link-Layer Threats: Eavesdropping

- Also termed ***sniffing***
- For subnets using broadcast technologies (e.g., WiFi, some types of Ethernet), get it for "free"
  - Each attached system's NIC (= Network Interface Card) can capture any communication on the subnet
  - Some handy tools for doing so
    - **`tcpdump`** (low-level ASCII printout)

`tcpdump`

- The **`tcpdump`** program allows you to see packets on the network
  - It puts your computer's card into promiscuous mode so it ignores MAC addresses
- You can add additional filters to isolate things
  - EG, only to and from your own IP
  - **`sudo tcpdump -i en0 host {myip}`**
- Note: this is ***wiretapping***
  - DO NOT RUN on a random open wireless network without a filter to limit the traffic you see
  - Only run without filters when connected to your own network
    - But do run it at home!

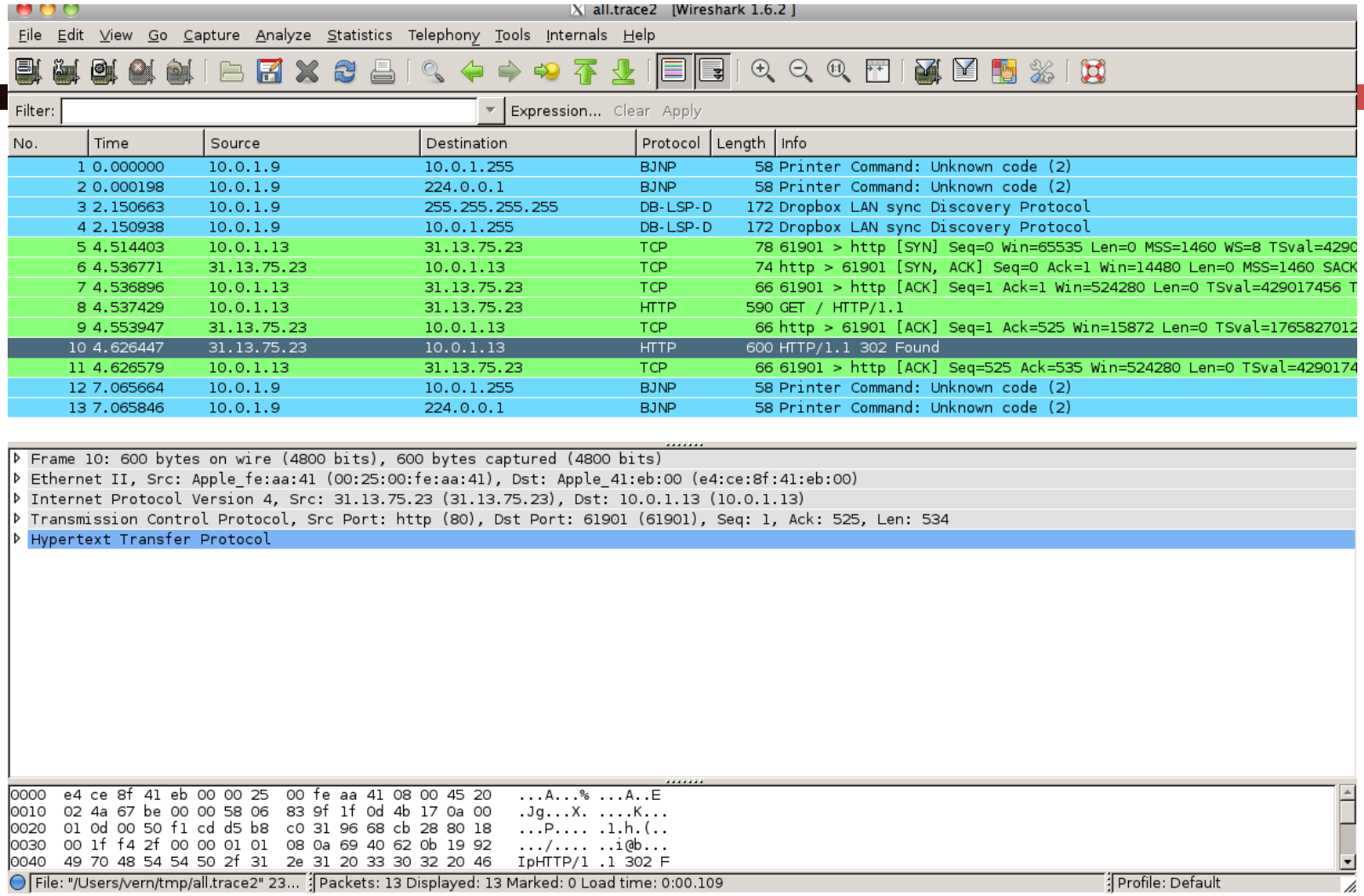
TCPDump


```
demo 2 % tcpdump -r all.trace2
reading from file all.trace2, link-type EN10MB (Ethernet)
21:39:37.772367 IP 10.0.1.9.60627 > 10.0.1.255.canon-bjnp2: UDP, length 16
21:39:37.772565 IP 10.0.1.9.62137 > all-systems.mcast.net.canon-bjnp2: UDP, length 16
21:39:39.923030 IP 10.0.1.9.17500 > broadcasthost.17500: UDP, length 130
21:39:39.923305 IP 10.0.1.9.17500 > 10.0.1.255.17500: UDP, length 130
21:39:42.286770 IP 10.0.1.13.61901 > star-01-02-pao1.facebook.com.http: Flags [S], seq 2
523449627, win 65535, options [mss 1460,nop,wscale 3,nop,nop,TS val 429017455 ecr 0,sack
OK,eol], length 0
21:39:42.309138 IP star-01-02-pao1.facebook.com.http > 10.0.1.13.61901: Flags [S.], seq
3585654832, ack 2523449628, win 14480, options [mss 1460,sackOK,TS val 1765826995 ecr 42
9017455,nop,wscale 9], length 0
21:39:42.309263 IP 10.0.1.13.61901 > star-01-02-pao1.facebook.com.http: Flags [.], ack 1
, win 65535, options [nop,nop,TS val 429017456 ecr 1765826995], length 0
21:39:42.309796 IP 10.0.1.13.61901 > star-01-02-pao1.facebook.com.http: Flags [P.], seq
1:525, ack 1, win 65535, options [nop,nop,TS val 429017456 ecr 1765826995], length 524
21:39:42.326314 IP star-01-02-pao1.facebook.com.http > 10.0.1.13.61901: Flags [.], ack 5
25, win 31, options [nop,nop,TS val 1765827012 ecr 429017456], length 0
21:39:42.398814 IP star-01-02-pao1.facebook.com.http > 10.0.1.13.61901: Flags [P.], seq
1:535, ack 525, win 31, options [nop,nop,TS val 1765827083 ecr 429017456], length 534
21:39:42.398946 IP 10.0.1.13.61901 > star-01-02-pao1.facebook.com.http: Flags [.], ack 5
```

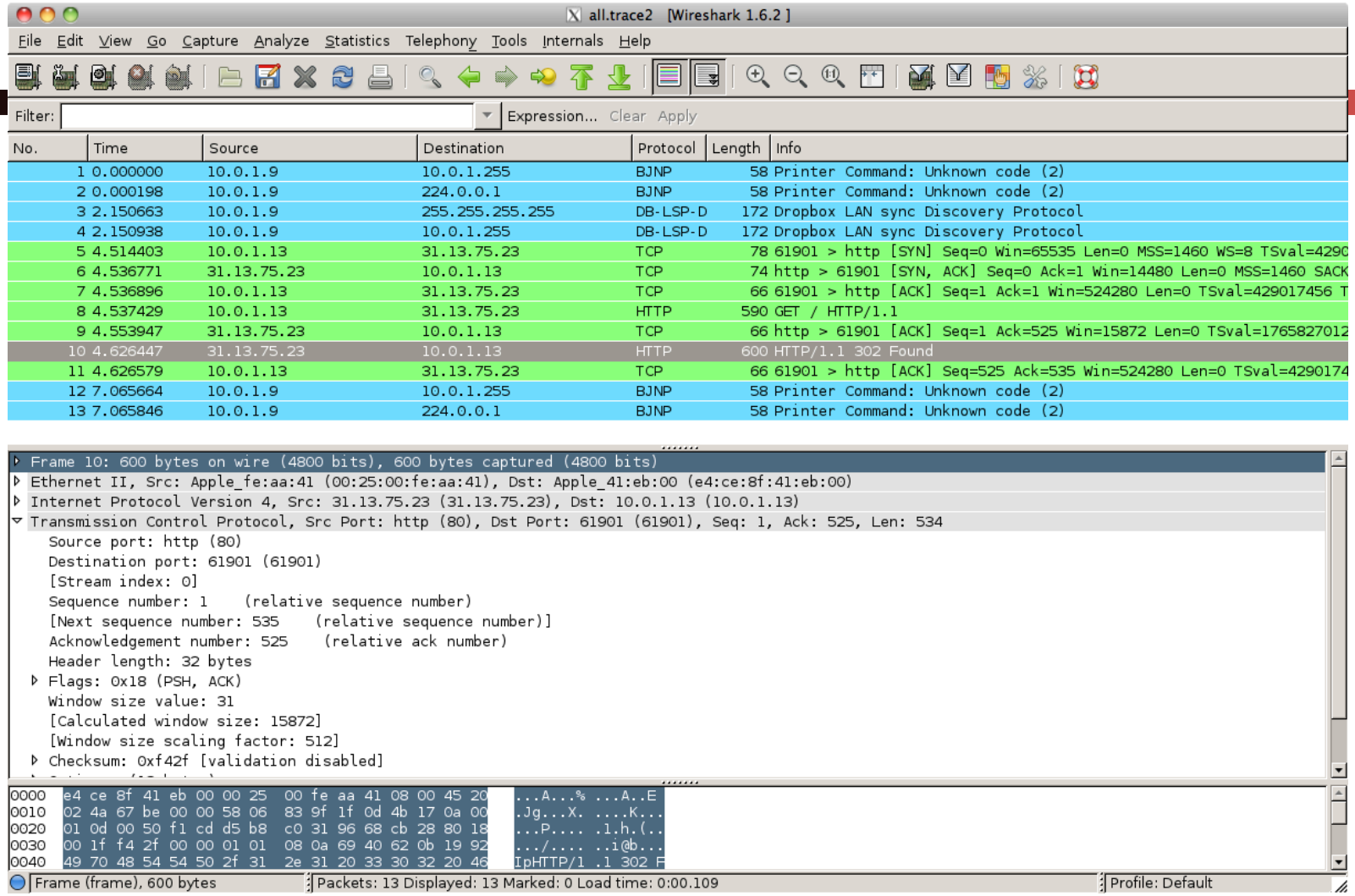
Physical/Link-Layer Threats: Eavesdropping

- Also termed ***sniffing***
- For subnets using broadcast technologies (e.g., WiFi, some types of Ethernet), get it for "free"
  - Each attached system's NIC (= Network Interface Card) can capture any communication on the subnet
  - Some handy tools for doing so
    - **`tcpdump`** (low-level ASCII printout)
    - Wireshark (higher-level printing)

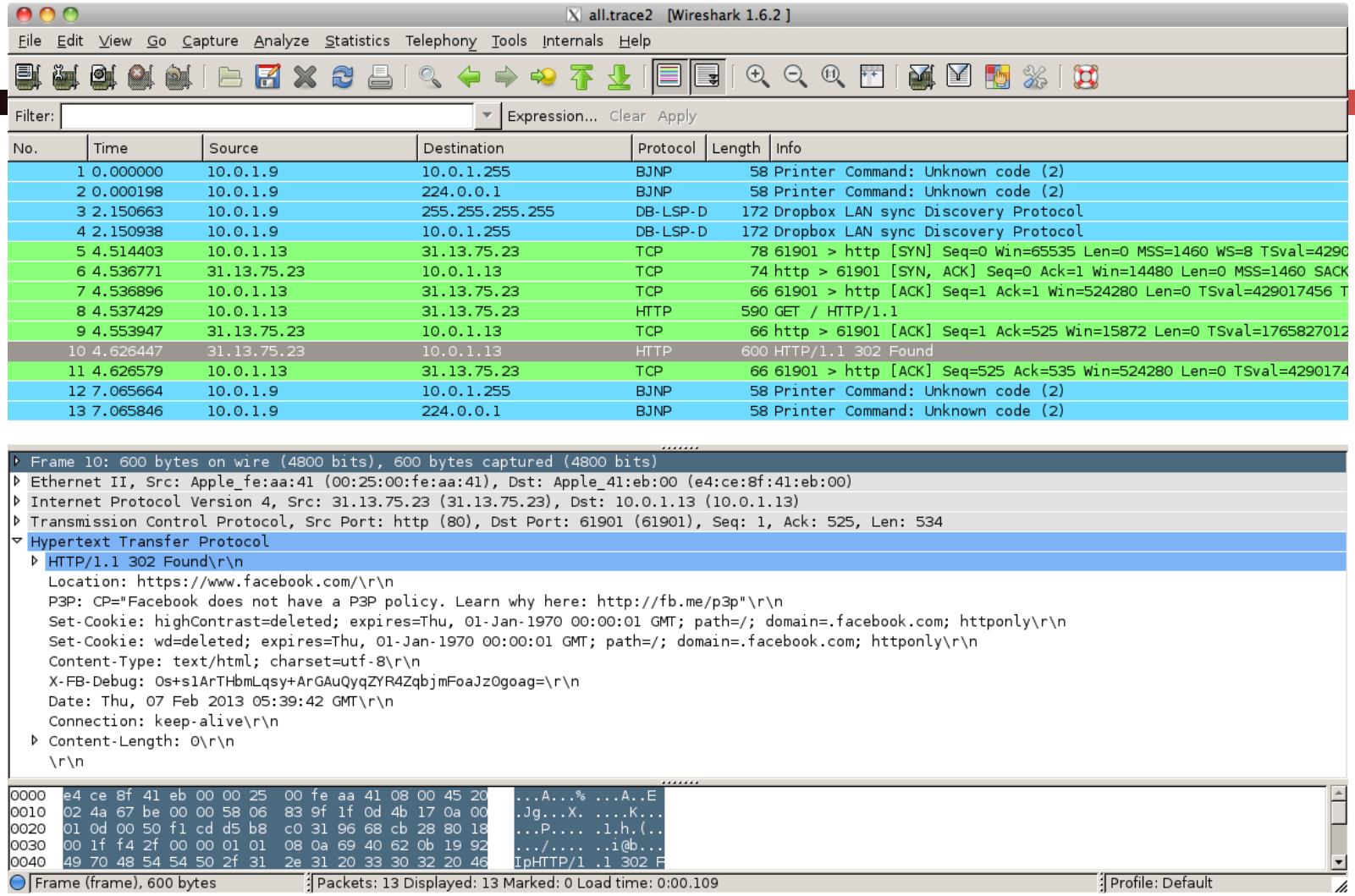
Wireshark: GUI for Packet Capture/Exam.



Wireshark: GUI for Packet Capture/Exam.



Wireshark: GUI for Packet Capture/Exam.

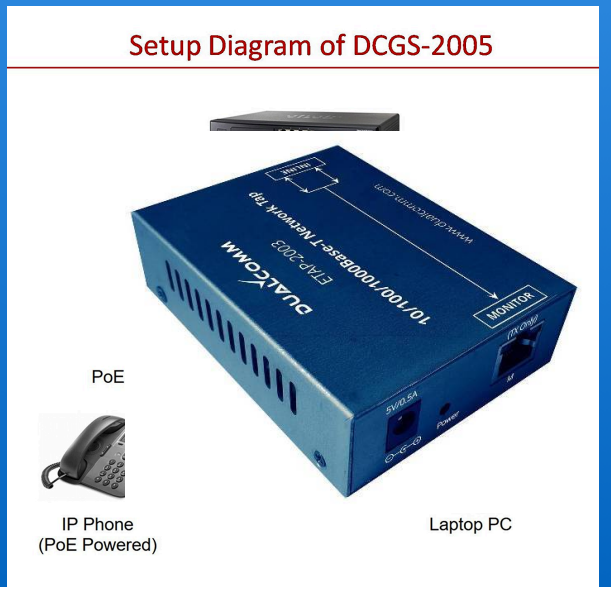


Physical/Link-Layer Threats: Eavesdropping

- Also termed ***sniffing***
- For subnets using broadcast technologies (e.g., WiFi, some types of Ethernet), get it for "free"
  - Each attached system's NIC (= Network Interface Card) can capture any communication on the subnet
  - Some handy tools for doing so
    - **`tcpdump`** (low-level ASCII printout)
    - Wireshark (higher-level printing)
    - zeek (scriptable real-time network analysis; see zeek.org)
- You can also "tap" (mirror) a link or configure a "mirror port"

One Of Nick's Favorite Toys: DualComm DCGS-2005

- A $200, 5-port Ethernet switch...
  - With some bonus features
- Built in port "mirror"
  - All traffic to and from port 1 is copied to port 5
- Powered through a USB connection
  - So no need for an extra power supply
- Power-Over-Ethernet passthrough
  - Port 2 can send power to port 1 so you can tap IP phones...
- Allows injecting packets as well:
  It is a switch
- No longer made but the same functionality is in the ETAP-2003R in a smaller package



Operation Ivy Bells

By Matthew Carle
Military.com

At the beginning of the 1970's, divers from the specially-equipped submarine, USS Halibut (SSN 587), left their decompression chamber to start a bold and dangerous mission, code named "Ivy Bells".

The Regulus guided missile submarine, USS Halibut (SSN 587) which carried out Operation Ivy Bells.

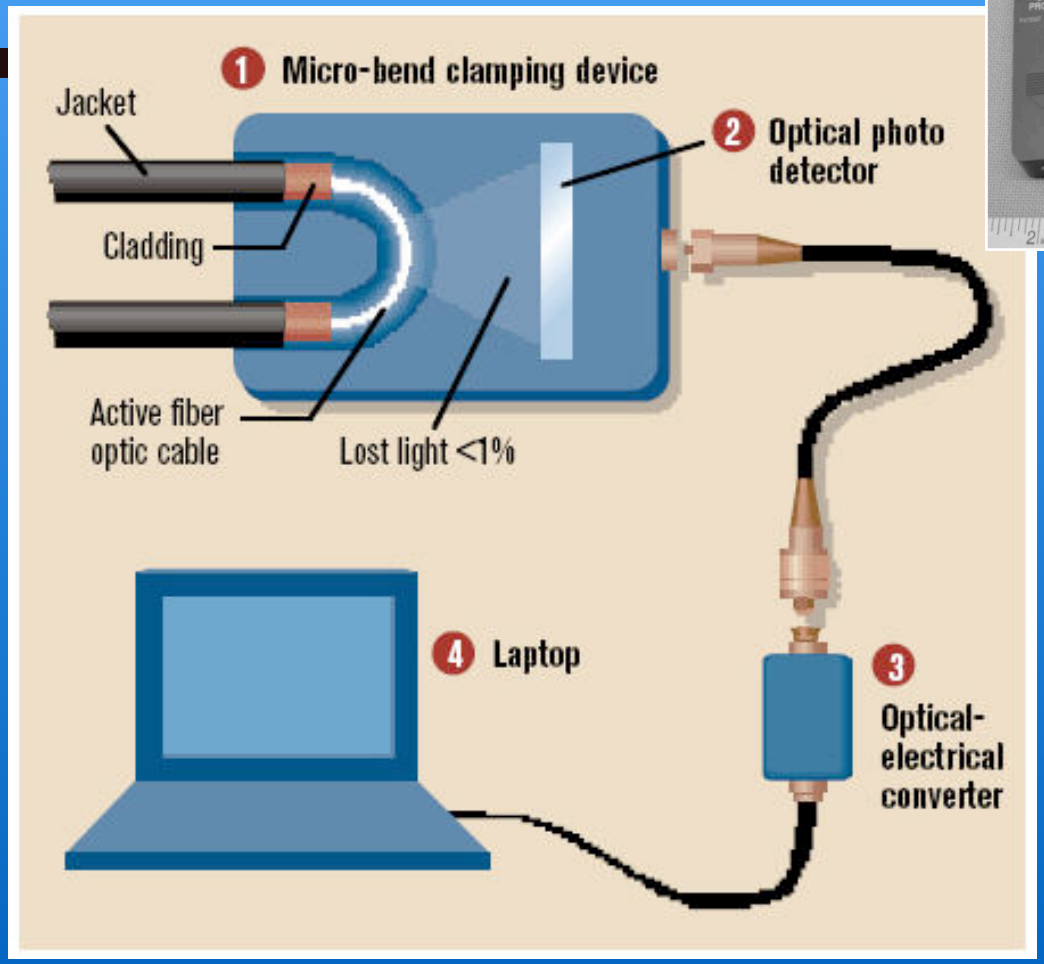
In an effort to alter the balance of Cold War, these men scoured the ocean floor for a five-inch diameter cable carry secret Soviet communications between military bases.

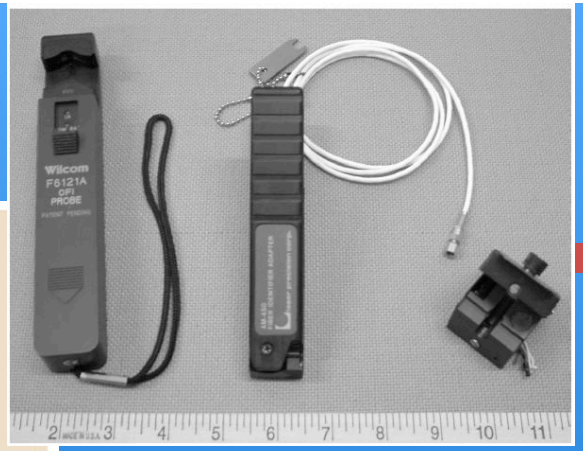
The divers found the cable and installed a 20-foot long listening device on the cable. designed to attach to the cable without piercing the casing, the device recorded all communications that occurred. If the cable malfunctioned and the Soviets raised it for repair, the bug, by design, would fall to the bottom of the ocean. Each month Navy divers retrieved the recordings and installed a new set of tapes.

Upon their return to the United States, intelligence agents from the NSA analyzed the recordings and tried to decipher any encrypted information. The Soviets apparently were confident in the security of their communications lines, as a surprising amount of sensitive information traveled through the lines without encryption.

prison. The original tap that was discovered by the Soviets is now on exhibit at the KGB museum in Moscow.

Stealing Photons





Types of Network Attackers...

- Off Path: 😑 I see NOTHING
  - Attacker is unable to see the network traffic of the victim
- On Path/Man On The Side: 🧐 I see you
  - Attacker can see packets...
  - Attacker can also add packets 📣
  - Attacker ***can not*** block legitimate packets
  - AKA "Nick with that DualComm toy"
- In Path/Man In The Middle: 🤬 I see you and can censor you
  - Attacker can see packets
  - Attacker can add packets
  - Attacker can block legitimate packets
    - Together the attacker can ***replace*** packets

The Basic Ethernet Packet: The near-universal Layer 2

- An Ethernet Packet contains:
  - A preamble to synchronize data on the wire
    - We normally ignore this when talking about Ethernet
  - 6 bytes of destination MAC address
    - In this case, MAC means media access control address, not message authentication code!
  - 6 bytes of source MAC address
  - Optional 4-byte VLAN tag
  - 2 bytes length/type field
  - 46-1500B of payload

The MAC Address

- The MAC acts as a device identifier
  - The upper 3 bytes are assigned to a manufacturer
    - Can usually identify product with just the MAC address
  - The lower 3 bytes are assigned to a specific device
    - Making the MAC a de-facto serial #
- Usually written as 6 bytes in hex:
  - e.g. `13:37:ca:fe:f0:0d`
- A device ***should ignore*** all packets that aren't to itself or to the broadcast address (`ff:ff:ff:ff:ff:ff`)
  - But almost all devices can go into ***promiscuous mode***
    - This is also known as "sniffing traffic"
- A device generally should only send with its own address
  - But this is enforced with software and can be trivially bypassed when you need to write "raw packets"

ARP: How To Find A System's Ethernet Address...

- We think of computers in terms of IP address...
  - But layer 2 doesn't know about IP addresses
- How to translate?
  - If its non-local (not in the same subnet), you just find the IP of the gateway
  - If it is on the same subnet (e.g same /24 == xx.yy.zz.?? are the same) find its address
- ARP is a broadcast protocol
  - Your computer shouts "Who has IP **`A.B.C.D`**" to the Ethernet broadcast address **`ff:ff:ff:ff:ff:ff`**
  - Computer that owns the address replies "IP **`A.B.C.D`** -> MAC **`aa:bb:cc:dd:ee:ff`**"

ARP Spoofing...

- A different computer decides to respond...
  - As long as it responds faster, it wins!
- Can be used to leverage to become a man-in-the-middle
  - See the ARP request for the gateway:
    Spoof the response
  - Now all traffic to the Internet instead goes to your system
- Requires seeing the ARP traffic so you have to be on the same local network

The Hub...

- In the old days, Ethernet was simply a shared broadcast medium
  - Every system on the network could hear every sent packet
  - How it implemented sharing:
    - If someone is talking, you don't talk
    - If nobody is talking, feel free to start
      - But if someone else starts talking at the same time, stop and do a random wait
- Implemented by either a long shared wire or a “hub” which repeated every message to all other systems on the network
  - Thus the only thing preventing every other computer from listening in is simply the network card’s default to ignore anything not directed at it
- The hub or wire is incapable of enforcing sender's MAC addresses
  - Any sender could simply lie about it’s MAC address when constructing a packet

The Hub Yet Lives!

- WiFi is effectively "Ethernet over Wireless"
  - With ***optional*** encryption which we will cover later
- Open (no password) wireless networks are just like the old Ethernet hub:
  - Any recipient can hear all the other sender's traffic
  - Any sender can use any MAC address it desires
- With the added bonus of easy to hijack connections
  - By default, your computer sends out "hey, is anyone here" looking for networks it knows
  - For open networks, anybody can say "Oh, yeah, here I am" and your computer connects to them

Rogue Access Points...

- Since unsecured wireless has no authentication...
  - And since devices by default shout out "hey, is anyone here network X"
- You can create an AP that simply responds with "of course I am"
  - The mana toolkit: https://github.com/sensepost/mana
- Now simply relay the victim's traffic onward
  - And do whatever you want to any unencrypted requests that either happen automatically or when the user actually does something
- Recommendations:
  - Do ***not*** remember unsecured networks
  - Do ***not*** have your computer auto-join open networks

Broadcast is Dangerous: Packet Injection

- If your attacker can see your packets…
  - It isn't just an information leakage
- Instead, an attacker can also ***inject*** their own packets
  - The low level network does not enforce any ***integrity or authenticity***
- So unless the high level protocol uses cryptographic checks…
- The target simply accepts the ***first*** packet it receives as valid!
  - This is a "race condition attack", whichever packet arrives first is accepted
  - This is conducted by an "on-path adversary", the adversary can see and add traffic but can't remove traffic

Packet Injection in Action: Airpwn


```
HTTP 302 FOUND
location: http://www.evil.com/hello.jpg
```

```
GET /foo/image.jpg HTTP/1.1
host: www.anydomain.com
```

```
GET /foo/image.jpg HTTP/1.1
host: www.anydomain.com
```

```
HTTP 200 OK
.....
```

```
HTTP 200 OK
....
Here's the goatee image
it will be seared into
your brain forever…
MUAHAHAHAHAHAHAH
```

But Airpwn ain't a joke…

- It is trivial to replace "look for .jpg request and reply with redirect to goatse" with "look for .js request and reply with redirect to exploitive javascript"
  - This JavaScript would start running in the target's web browser, profile the browser, and then use whatever exploits exist
- The requirements for such an attack:
  - The target's traffic must not be encrypted
  - The ability to see the target's traffic
  - The ability to determine that the target's traffic belongs to the target
  - The ability to inject a malicious reply

So Where Does This Occur?

- Open wireless networks
  - E.g. Starbucks, and ***any wireless network without a password***
  - Only safe solution for open wireless is ***only*** use encrypted connections
    - HTTPS/TLS, **`ssh`**, or a Virtual Private Network to a better network
- On backbones controlled by nation-state adversaries!
  - The NSA's super-duper-top-secret attack tool, QUANTUM is ***literally*** airpwn without the goatse!
    - Not an exaggeration: Airpwn only looks at single packets, so does QUANTUM!

It's also ***too*** easy

- Which is why it isn't an assignment!
- Building it in scapy, a packet library in python:
  - Open a sniffer interface in one thread
    - Pass all packets to a separate work thread so the sniffer doesn't block
  - For the first TCP data packet on any flow destined on port 80
    - Examine the payload with a simple regular expression to see if its a fetch for an image (ends in .jpg or .gif) and not for our own server
      - Afterwards whitelist that flow so you ignore it
  - If so, construct a 302 reply
    - Sending the browser to the target image
  - And create a fake TCP packet in reply
    - Switch the SYN and ACK, ports, and addresses
    - Set the ACK to additionally have the length of the request
    - Inject the reply
    - And send it with Connection: close and terminate the TCP connection

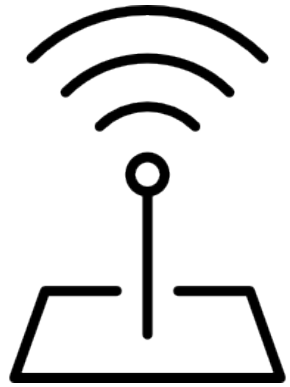
Detecting Injected Packets: Race Conditions

- Clients ***can*** detect an injected packet
  - Since they still see the original reply
- Packets can be duplicated, but they should be consistent
  - EG, one version saying “redirect”, the other saying “here is contents” should not occur and represents a ***necessary*** signature of a packet injection attack
- Problem: often detectable too late
  - Since the computer may have acted on the injected packet in a dangerous way before the real reply arrives
- Problem: nobody does this in practice
  - So you don't actually see the detectors work
- Problem: “Paxson’s Law of Internet Measurement”
  - “The Internet is weirder than you think, even when you include the effects of Paxson’s Law of Internet Measurement”
  - Detecting bad on the Internet often ends up inadvertently detecting just odd:
    Things are always more broken then you think they are

Wireless Ethernet Security Option: WPA2 Pre Shared Key

- This is what is used these days when the WiFi is “password protected”
  - The access point and the client have the same pre-shared key (called the PSK key)
  - Goal is to create a shared key called the PTK (Pairwise Transient Key)
- This key is derived from a combination of both the password and the SSID (network name)
  - PSK = PBKDF2(passphrase, ssid, 4096, 256)
- Use of PBKDF
  - The SSID as salt ensures that the same password on different network names is different
  - The iteration count assures that it is ***slow***
    - Any attempt to brute force the passphrase should take a lot of time per guess

The WPA 4-way Handshake



SNonce + MIC

Computed PTK = F(PSK, ANonce SNonce, AP MAC, Client MAC)

ANonce

GTK + MIC

Computed PTK = F(PSK, ANonce SNonce, AP MAC, Client MAC)

Remarks

- This is ***only*** secure if an eavesdropper doesn't know the pre shared key
  - Otherwise an eavesdropper who sees the handshake can perform the same computations to get the transport key
    - However, by default, network cards don't do this:
      This is a "do not disturb sign" security. It will keep the maid from entering your hotel room but won't stop a burglar
  - Oh, and given ANonce, SNonce, MIC(SNonce), can attempt an ***off line*** brute-force attack
- The MIC is really a MAC, but as MAC also refers to the MAC address, they use MIC in the description
- The GTK is for broadcast
  - So the AP doesn't have to rebroadcast things, but usually does anyway

Rogue APs and WPA2-PSK...

- You can ***still do a rogue AP***!
  - Just answer with a random ANonce...
  - That gets you back the SNonce and MIC(SNonce)
    - Which uses as a key for the MIC = F(PSK, ANonce, SNonce, AP MAC, Client MAC)
- So just do a brute-force dictionary attack on PSK
  - Since PSK = PBKDF2(pw, ssid, 4096, 256)
  - Verify the MIC to validate whether the guess was correct
- Because lets face it, people don't chose very good passwords...
  - You could probably build a nice one on a Jetson Nano dev-board:
    Linux computer with a 1/2 TFlop GPU in a $100 package

Actually Making it Secure: WPA Enterprise

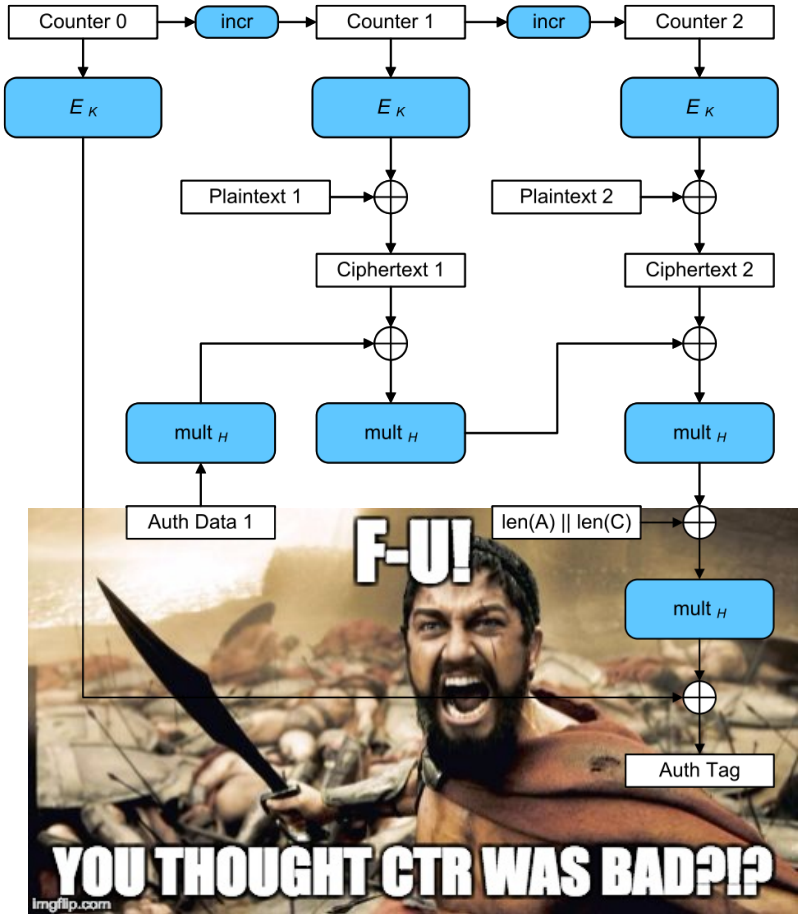
- When you set up Airbears 2, it asks you to accept a public key certificate
  - This is the public key of the authentication server
- Now ***before*** the 4-way handshake:
  - Your computer first handshakes with the authentication server
    - This is secure using public key cryptography (TLS which we discuss later)
  - Your computer then authenticates to this server
    - With your username and password
- The server now generates a unique key instead of the PSK that it both tells your computer and tells the base station over encrypted channels
  - So the 4 way handshake is now secure

Recent Hotness: KRACK attack...

- To actually encrypt the individual packets: IV of a packet is {Agreed IV || packet counter}
  - Thus for each packet you only need to send the packet counter (48 bits) rather than the full IV (128b)
- Multiple different modes
  - One common one is CCM (Counter with CBC-MAC)
    - MAC the data with CBC-MAC
      Then encrypt with CTR mode
  - The highest performance is GCM (Galois/Counter Mode)
- But if you thought CTR mode was bad on IV reuse...
  - GCM is worse: A couple of reused IVs can reveal enough information to forge the authentication!
- Discovered a couple years ago, fairly quickly patch, but...

GCM...

- GCM is like CTR mode with a twist...
  - The confidentiality is pure CTR mode
  - The "Galois" part is a hash of the cipher text
    - The only secret part being the "Auth Data"
- Reuse the IV, what happens?
  - Not ***only*** do you have CTR mode loss of confidentiality...
  - But now you can get the counter-0 for the auth tag...
  - So you lose the integrity that GCM supposedly provided!



And Packets Get "Lost"

- Even a wired network will "drop packets"
  - A message is sent but simply never delivered
- Its far worse on wireless
  - A gazillion things can go wrong, including other transmitters
    - And noise like a microwave oven!
- So you have to design for packets to be rebroadcast...
- In the WPA handshake, what do you do when you receive the 3rd packet?
  - Initialize the key you use for encrypting the packets
  - Set the packet counter to 0

And A Replay Attack...

- What if the attacker listens for the third step in the handshake...
  - And then repeats it?
- Why, the client is supposed to reinitialize the key and agreed IV...
  - Which on many implementations, ***also resets the packet counter***...
  - Oh, and Linux (and Android 6) was worse... It reinitializes the key ***to zero!***
- So what does that mean?

Attack Scenario...

- Attacker is close to target
- Attacker captures the 3rd step in the handshake
- Attacker repeatedly replays this to the client
- Client now repeats IVs for encryption...
- Other modes. Annoyance: the damage is minor
- CCM-mode: Attacker can now decrypt in practice thanks to IV reuse
- GCM-mode...
  - Attacker can now decrypt ***and forge packets***:
    Reusing the IV also reveals the MAC-secret!

Mitigations...

- Like all attacks on WiFi, it requires a "close" attacker...
  - 100m to a km or two...
- If you use WPA2-PSK, aka a "WiFi Password", who cares?
  - Unless your WiFi password sounds like a cat hawking up a hairball, you don't have enough entropy to resist a brute-force attacks
- If you use WPA2-Enterprise, this ***may*** have mattered...
  - But lets face it, there are so many more critical things to patch first...
  - And why are you treating the WiFi as trusted anyway?
- And it did get patched quickly

Dragonfly…

- WPA2-PSK sucks…
  - An eavesdropper or a rogue access point gains enough information for an ***offline*** attack on the pre-shared password
- What we really want is “Simultaneous Authentication of Equals”
  - Alice and Bob share the same password ***PW***
  - Alice and Bob can negotiate a shared public-key based secret only if both know ***PW***
  - If Alice or Bob doesn’t know ***PW***, then they don’t learn anything about ***PW*** unless they successfully guessed ***PW*** during the protocol
    - Both Alice and Bob know the password in the clear
- Enter Dragonfly (RFC 7664)
  - Has both EC and conventional DH based variants

Last part is important:
Turns offline into online-only attacks

- If Alice or Bob doesn't know ***PW***, then they don't learn anything about ***PW*** *unless they successfully guessed* ***PW*** during the protocol
  - Model is we have a set of possible passwords, all equally likely, which one?
- Off-line attacks are death:
  - Attacker can try as many passwords as they want in parallel
- On-line attacks are much more limited:
  - Attacker can only try one at a time...
  - And can rate-limit the attacker
- iOS passcode design is strongly set up to force online-only attacks: Even if you compromise the secure enclave, you have to try each password sequentially

DH-based Dragonfly

- Public parameters:
  - A prime ***p***
    - A generator over this ***G***
  - A (smaller) prime ***q***
    - Size of the group defined by ***G*** and ***q*** is a large prime divisor of ***(p-1)/2***
  - A selected generator ***g*** is valid if ***g*** < ***p*** and $g^q \bmod p = 1$
  - Same idea as with DSA: We can use a smaller specialized group and be sending smaller data elements around
- Identifiers for Alice and Bob
  - EG, MAC addresses, with an ordering function
- Key idea:
  - Select a ***random*** generator ***g,*** called ***P*** (or ***PE*** == Password Element) based on $H(ID_a \| ID_b \| PW)$

Actually creating PE


```
found = False
counter = 1
n = len(p) + 64
do {
  base = H(max(Alice,Bob) | min(Alice,Bob) | password | counter)
  temp = KDF-n(base, "Dragonfly Hunting And Pecking")
  seed = (temp mod (p - 1)) + 1
  temp = seed ^ ((p-1)/q) mod p
  if (temp > 1)
  then
    if (not found)
      PE = temp
      found = true
    fi
  fi
  counter = counter + 1
} while ((!found) || (counter <= k))
```

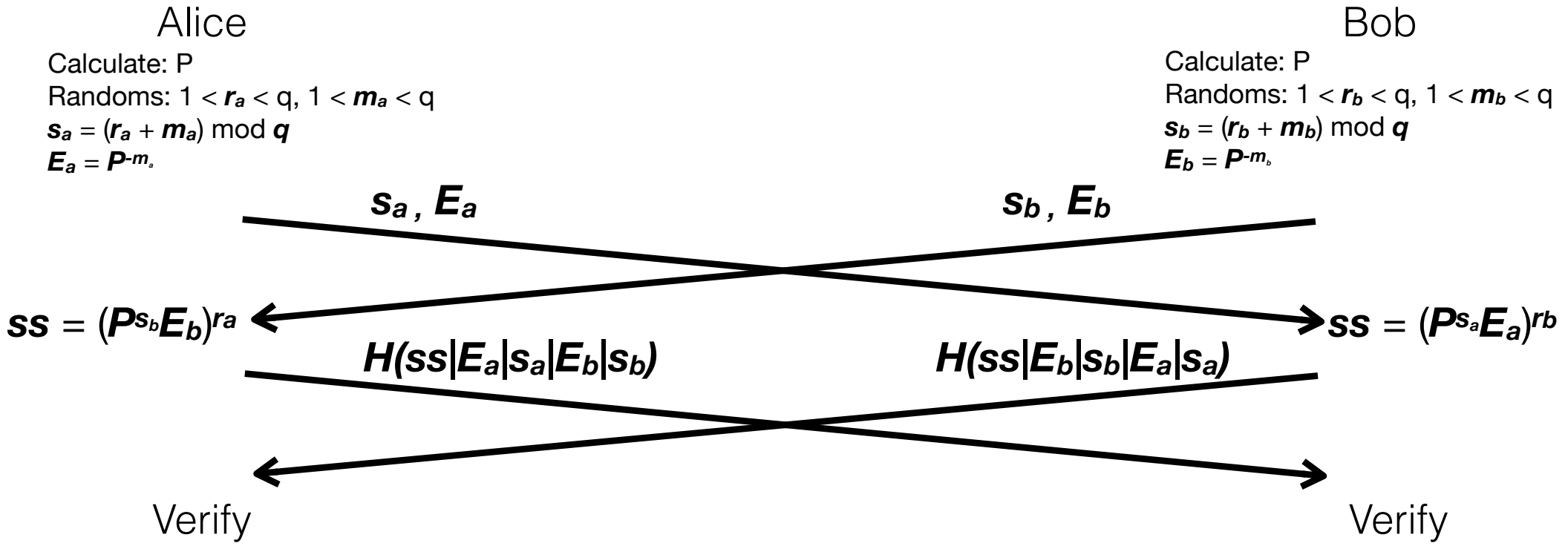
Remarks…

- Called “Hunting and pecking”:
  - Select a (pseudo)-Random element, check if its valid
  - If not, repeat
- We need this to resist side-channel attacks
  - So we specify a minimum iteration count ***k***
  - We select the first one, but we keep at it for a suitable ***k*** so the probability of failure is low enough: but still often 40+ times!
- We can’t precompute this because we include Alice and Bob’s identity in determining ***P***
  - Eliminating this would eliminate the need for online computation of ***P***
  - But we can cache ***P*** still: an important optimization since this calculation is expensive!

Now to prove that everybody knows the same P... And generate a key

- Alice creates two random values:
  - $1 < r_a < q$ (the random value)
  - $1 < m_a < q$ (the mask value)
- Alice now computes
  - $s_a = (r_a + m_a) \bmod q$
  - $E_a = P^{-mask}$
  - Sends those to Bob, Bob sends his counterparts to Alice
- Now the starting secret…
- $ss = (P^{s_b} E_b)^{r_a} = (P^{(r_b + m_b - m_b)})^{r_a} = P^{r_a r_b}$
- Sends those to Bob, Bob sends his counterparts
- Verify P^{s_b} and Sb are valid
- Computes $H(ss|E_a|s_a|E_b|s_b)$ and sends that to Bob
- verifies Bob's counterpart which uses a different order
- Final:
  $K = H(ss|E_a * E_b|s_a + s_b)$

Graphically



$$K = H(ss|E_a * E_b|s_a + s_b)$$

Use in WPA3

- WPA3 does this
  - Well, over an elliptic curve instead, but same idea:
    Generate a random generator and use that
- But it is not during the 4-way handshake…
  - Instead, it is 2 additional handshakes ***before*** the 4-way handshake
  - Result is higher latency but, eh, 🤷
- Exists correctness and security proofs
- Result is WPA3:
  - ***Eliminates*** the off-line brute force attacks
  - ***Eliminates*** the "adversary with the password" L2 attacks

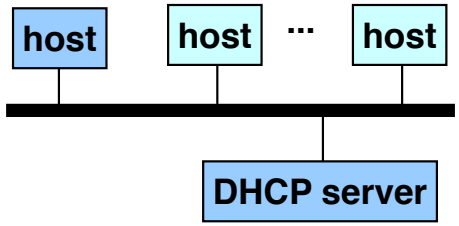
Broadcast Protocols Make The Local Network Worse...

- By default, both DHCP and ARP broadcast requests
  - Sent to ***all*** systems on the local area network
- DHCP: Dynamic Host Control Protocol
  - Used to configure all the important network information
    - Including the DNS server:
      If the attacker controls the DNS server they have complete ability to intercept all traffic!
    - Including the Gateway which is where on the LAN a computer sends to:
      If the attacker controls the gateway
- ARP: Address Resolution Protocol
  - "Hey world, what is the Ethernet MAC address of IP X"
  - Used to find both the Gateway's MAC address and other systems on the LAN

2. Configure your connection

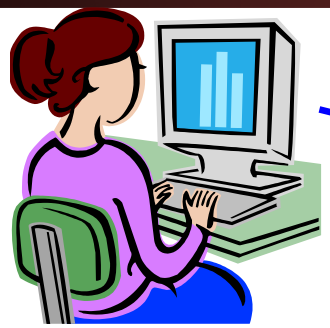
Internet Bootstrapping: DHCP

- New host doesn't have an IP address yet
  - So, host doesn't know what source address to use
- Host doesn't know *who to ask* for an IP address
  - So, host doesn't know what destination address to use
- (Note, host does have a separate WiFi MAC address)
- Solution: *shout* to "**discover**" server that can help
  - Broadcast a server-discovery message (layer 2)
  - Server(s) sends a reply offering an address



DHCP = Dynamic Host Configuration Protocol

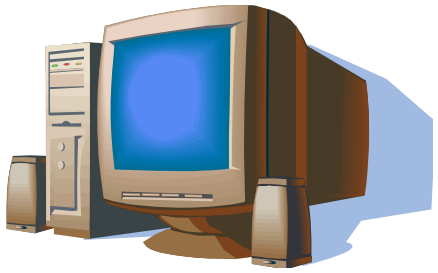
Dynamic Host Configuration Protocol



new client

DHCP discover (broadcast)

DHCP offer



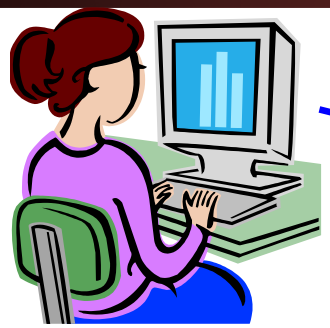
DHCP server

"**offer**" message includes IP address, DNS server, "gateway router", and how long client can have these ("lease" time)

DNS server = system used by client to map hostnames like `gmail.com` to IP addresses like `74.125.224.149`

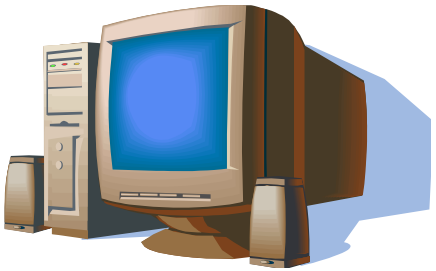
Gateway router = router that client uses as the first hop for all of its Internet traffic to remote hosts

Dynamic Host Configuration Protocol



new client

DHCP discover (broadcast)



DHCP server

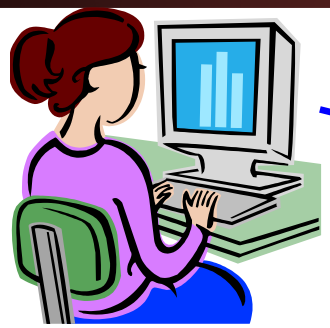
DHCP offer

"**offer**" message includes IP address, DNS server, "gateway router", and how long client can have these ("lease" time)

DHCP request (broadcast)

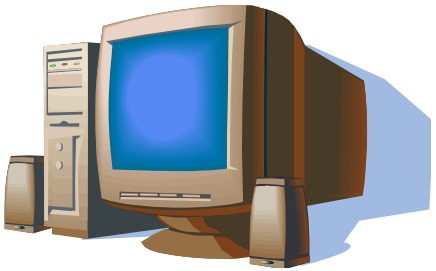
DHCP ACK

Dynamic Host Configuration Protocol



new client

DHCP discover (broadcast)



DHCP server

DHCP offer

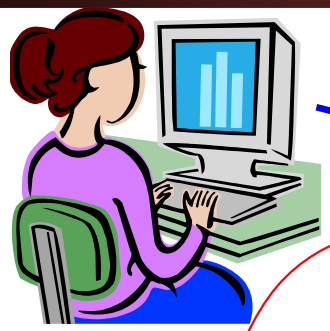
"**offer**" message includes IP address, DNS server, "gateway router", and how long client can have these ("lease" time)

DHCP request (broadcast)

Threats?

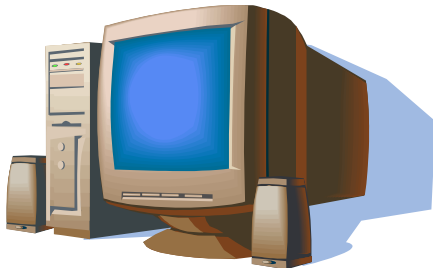
DHCP ACK

Dynamic Host Configuration Protocol



new client

DHCP discover (broadcast)



DHCP server

DHCP offer

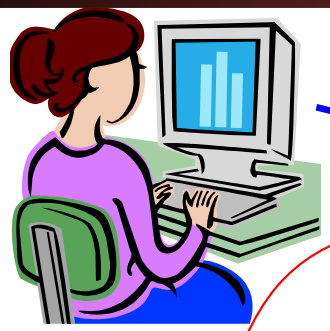
"**offer**" message includes IP address, DNS server, "gateway router", and how long client can have these ("lease" time)

DHCP request (broadcast)

Local attacker on same subnet can **hear** new host's DHCP request

DHCP ACK

Dynamic Host Configuration Protocol



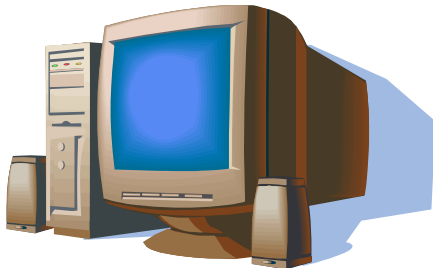
new client

DHCP discover (broadcast)

DHCP offer

DHCP request (broadcast)

DHCP ACK



DHCP server

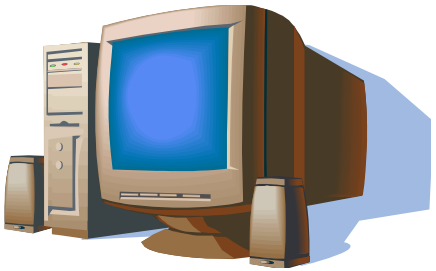
"**offer**" message includes IP address, DNS server, "gateway router", and how long client can have these ("lease" time)

This happens **even for WPA2-Enterprise**, since request is explicitly sent using broadcast

Dynamic Host Configuration Protocol

new client

DHCP discover (broadcast)



DHCP server

DHCP offer

"**offer**" message includes IP address, DNS server, "gateway router", and how long client can have these ("lease" time)

DHCP request (broadcast)

DHCP ACK

Attacker can **race** the actual server; if attacker wins, replaces DNS server and/or gateway router

DHCP Threats

- Substitute a fake DNS server
  - Redirect any of a host's lookups to a machine of attacker's choice (e.g., **`gmail.com`** = **`6.6.6.6`**)
- Substitute a fake gateway router
  - Intercept all of a host's off-subnet traffic
  - Relay contents back and forth between host and remote server
    - Modify however attacker chooses
  - This is one type of invisible Man In The Middle (MITM)
    - Victim host generally has no way of knowing it's happening! 😟
    - (Can't necessarily alarm on peculiarity of receiving multiple DHCP replies, since that can happen benignly)
- How can we fix this?

Hard, because we lack a ***trust anchor***

DHCP Conclusion

- DHCP threats highlight:
  - Broadcast protocols inherently at risk of local attacker spoofing
    - Attacker knows exactly when to try it …
    - … and can see the victim's messages
  - When initializing, systems are particularly vulnerable because they can lack a trusted foundation to build upon
  - Tension between ***wiring in trust*** vs. ***flexibility and convenience***
  - MITM attacks insidious because no indicators they're occurring

So How Do We Secure the LAN?

- Option 1: We don't
  - Just assume we can keep bad people out
  - This is how most people run their networks:
    "Hard on the outside with a goey chewy caramel center"
- Option 2: ***smart*** switching and active monitoring

The Switch

- Hubs are very inefficient:
  - By broadcasting traffic to all recipients this greatly limits the aggregate network bandwidth
- Instead, most Ethernet uses switches
  - The switch keeps track of which MAC address is seen where
- When a packet comes in:
  - If it is to the broadcast address, send it to all ports
  - If there is no entry in the MAC cache for the destination, broadcast it to all ports
  - If there is an entry, send it just to that port
- Result is vastly improved bandwidth
  - All ports can send or receive at the same time

Smarter Switches: Clean Up the Broadcast Domain

- Modern high-end switches can do even more
  - A large amount of potential packet processing on items of interest
- Basic idea: constrain the broadcast domain
  - Either filter requests so they only go to specific ports
    - Limits other systems from listening
  - Or filter replies
    - Limits other systems from replying
- Locking down the LAN is very important practical security
  - This is ***real*** defense in depth:
    Don't want 'root on random box, pwn whole network'
  - This removes "***pivots***" the attacker can try to extend a small foothold into complete network ownership
- This is why an Enterprise switch may cost $1000s yet provide no more real bandwidth than a $100 Linksys.

Smarter Switches:
Virtual Local Area Networks (VLANs)

- Our big expensive switch can connect a lot of things together
  - But really, many are in ***different*** trust domains:
    - Guest wireless
    - Employee wireless
    - Production desktops
    - File Servers
    - etc...
- Want to isolate the different networks from each other
  - Without actually buying separate switches

VLANs

- An ethernet port can exist in one of two modes:
  - Either on a single VLAN
  - On a trunk containing multiple specified VLANs
- All network traffic in a given VLAN stays only within that VLAN
  - The switch makes sure that this occurs
- When moving to/from a trunk the VLAN tag is added or removed
  - But still enforces that a given trunk can only read/write to specific VLANs

Putting It Together:
If I Was In Charge of UC networking...

- I'd isolate networks into 3+ distinct classes
  - The plague pits (AirBears, Dorms, etc)
  - The mildly infected pits (Research)
  - Administration
- Administration would be locked down
  - Separate VLANs
  - Restricted DHCP/system access
  - Isolated from the rest of campus