ICS 351: Today's plan

- IPv6 routing protocols (summary)
- HTML
- HTTP
- web scripting languages
- certificates (review)
- cookies

IPv6 routing

- almost the same routing protocols as for IPv4:
  - RIPng, OSPFv6, BGP with multiprotocol extensions
- more bits for the netmask, so more opportunities for subnetting
- plenty of (re)configuration!
  - but most of it automated

HTML

- HyperText Markup Language
- an in-line way of marking (hyper)text, similar in spirit to TeX/LaTeX, and inspiring the creation of XML
- part of the markings are about style and formatting: font, size, bold/italic, bullet lists, etc.
- some markings lead you to other pages or objects, e.g.

  home page, or

  <img src="http://www2.hawaii.edu/~esb/pix/2009sils.jpg">

  objects are identified by URLs (all URLs are also URIs)
- each URL has a protocol (scheme name, e.g. http), a host identifier (DNS name or IP address), an optional port number (:80 if not specified), and the path given to the server

typical HTTP interaction

- client is given a URL, splits it into domain name (port) and path
- client resolves domain name to IP address
- client opens a connection to the IP address (port 80, or the given port), server accepts connection (TCP 3-way handshake)
- client sends HTTP request
- server sends HTTP response
- after parsing response and finding embedded images or other content, client sends new HTTP requests on same TCP connection
- server replies to each request in sequence
- client matches each response to its request, renders the page
- after a time (typically 30s), the server closes the connection

HTTP request header

- all HTTP is rendered using ASCII. This makes it easy to read, a little harder to parse
- for example, an HTTP request might look like this:

```
GET /~esb/ HTTP/1.1
Host: www2.ics.hawaii.edu
Accept: */*
Connection: close
```

HTTP response header

- a corresponding HTTP reply might look like this:

```
HTTP/1.1 200 OK
Date: Thu, 19 Nov 2009 05:18:56 GMT
Server: Apache
Last-Modified: Wed, 02 Sep 2009 03:17:30 GMT
ETag: "19abf-2095-4728fb5090680"
Accept-Ranges: bytes
Content-Length: 8341
Connection: close
Content-Type: text/html

<html>...
```

HTTP headers

- in each case, the first line describes the main request or result:
  - in the request, the method can be GET, HEAD, POST, or a few others,
  - the path is specified immediately after the request,
  - the protocol version follows the path
  - in the reply, the version comes first, followed by the result code, both as a number and as a string
- the remaining lines of the header give more details, sometimes essential details (e.g. the content type and content length)
- each header ends with an empty line

HTTP/2

- headers are not ASCII, and support compression of header information
- server can push data that was not requested, for example images the server knows will be needed to render a web page
- content for several requests can be interleaved on a single TCP connection
  - slow content that the server begins to send early need not block later fast content

web scripting languages

- web content described by HTML was originally static, corresponding to files on the server
- since the server is a program, it can generate content dynamically, e.g. put the user's name (or bank balance) within the web page
- however, this would require modifying the code of the server
  - which is error-prone and hard to do
- so instead, the server program can execute a *server-side script* to generate new content to be served
- this script can be written in any language supported by the system on which the server is running

client-side scripts

- even with a server-side script, each change in the web page requires an HTTP request and reply, and requires that the page be rendered again
  - HTTP requests and replies can be slow
- usually also requires a mouse click
- to have more interactivity, many browsers have been designed to execute *client-side scripts* that can modify the displayed page
  - they may fetch data from the server
- client-side scripts are in Java or (now) Javascript

client-side scripts and security

- while client-side scripts do much to improve the appearance of pages, there can be concerns about security and reliability
- client-side scripts let servers execute code on a client – how does the client know what the code will do? can the client trust the server?
- in an attempt to address these concerns, browsers limit what scripts are allowed to do
- not all browsers execute client-side scripts

server-side scripts and security

- bugs in a server-side script can be exploited by attackers
- server-side scripts that do not thoroughly check their input are vulnerable, e.g. to SQL injection attacks

  http://xkcd.com/327/

- a server-side script lets the client execute code on the server
- the server controls what scripts are available, but not what the clients will do with the scripts

secure HTTP

- HTTP by itself is very insecure: any man-in-the-middle attacker can observe all the content sent and received
- some people wish to use HTTP to send sensitive data, e.g. credit card numbers, personal email
- instead of layering HTTP over TCP, HTTP can be layered over a secure protocol that runs over TCP
- the choice of secure protocols for HTTPS (secure HTTP) is SSL (older) or TLS (newer)

certificates

- a *certificate* is a digital signature by entity CA verifying that the enclosed public key authenticates server S
- there are a few (~100) certificate authorities (CAs) that are widely known and recognized by many web browsers
- when presenting its public key, a server S also presents the certificate signed by a CA as evidence that S indeed is the server the user wants to talk with

certificate vulnerabilities

- certificates protect against man-in-the-middle attack (including DNS attacks), but are still vulnerable to misspellings (e.g. `goggle.com`)
- if the certificate authority is compromised, and DNS or the routing infrastructure subverted, an attacker can impersonate any website
- this may have happened – the dutch CA diginotar may have had its keys stolen and misused

self-signed certificates

- if I have a website for private use, I don’t need a certificate from a CA
- I can use a *self-signed* certificate instead
- as before, the crucial step is giving the browser the correct public key for the desired server
- this requires hand-configuration of all the browsers that will use this server

HTTP cookies

- HTTP is a stateless protocol: a server has no real way to identify a client, so a request may or may not be connected with prior requests
- instead, a server may offer a client a *cookie*, a small amount of data that is only meaningful to the server
- on subsequent related requests to the same server, the client will send back the cookie, to confirm that the requests are connected
- cookies have an expiration time -- most cookies used for authentication expire quickly

HTTP cookies

- cookies can also be used to attempt to track users as they visit multiple sites, by embedding in the several sites a small image (or other content) served from the same server
- these cookies are often long-lived
- similar tracking can be done by tracking accesses based on the IP number of the connecting client

Cookie Persistence

```
HTTP/1.1 200 OK
Date: Sun, 06 Apr 2014 01:22:44 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Set-Cookie:
PREF=ID=ef4f230aa811ea46:FF=0:TM=1396747364:LM=1396747364:S=MKk0H_sL
4n4ASDWT; expires=Tue, 05-Apr-2016 01:22:44 GMT; path=/;
domain=.google.com
Set-Cookie: NID=67=JDP6w2jg7bqqHpOm0D6MNfqUwjiuH7YDQ_oGL3J-xt93-
BLfL4xjxVBEN-aTJ
NwX4nx6cRd9oVyTlHrPBi1XyZmEaWh3VHW3clsVNEIBjT2RA1h8mdWYQxcQr10-Nqnz;
expires=Mon
, 06-Oct-2014 01:22:44 GMT; path=/; domain=.google.com; HttpOnly
P3P: CP="This is not a P3P policy! See
http://www.google.com/support/accounts/bi
n/answer.py?hl=en&answer=151657 for more info."
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Alternate-Protocol: 80:quic
Connection: close
```

DNS reminder

- DNS provides name to IP address resolution
- Domain names are grouped into zones
- a DNS server provides translation (resolution) for the names in one zone
- a DNS query contains *question* Resource Records
- a DNS response may contain *answer* RRs, *name server* RRs, and *additional* RRs

```
dig hawaii.edu

;; QUESTION SECTION:
;hawaii.edu.                    IN    A

;; ANSWER SECTION:
hawaii.edu.             1800IN    A     128.171.224.100

;; AUTHORITY SECTION:
hawaii.edu.             1800IN    NS    dns4.hawaii.edu.
hawaii.edu.             1800IN    NS    dns2.hawaii.edu.
hawaii.edu.             1800IN    NS    dns1.hawaii.edu.

;; ADDITIONAL SECTION:
dns1.hawaii.edu.        1800IN    A     128.171.3.13
dns1.hawaii.edu.        1800IN    A     128.171.1.1
dns2.hawaii.edu.        1800IN    A     128.171.3.13
dns2.hawaii.edu.        1800IN    A     128.171.1.1
dns4.hawaii.edu.        1800IN    A     130.253.102.4
```

```
dig mx hawaii.edu
;; QUESTION SECTION:
;hawaii.edu.          IN MX
;; ANSWER SECTION:
hawaii.edu.      1800 IN MX   10 mx1.hawaii.edu.
;; AUTHORITY SECTION:
hawaii.edu.      1800 IN NS   dns1.hawaii.edu.
hawaii.edu.      1800 IN NS   dns4.hawaii.edu.
hawaii.edu.      1800 IN NS   dns2.hawaii.edu.
```

system administration

- suppose a system administrator has to manage a large number of machines
- for example, three web servers, a DHCP server, a backup server, a Network Attached Storage (NAS) server, a mail server, and a few printers
- a large KVM might be useful, but also has limitations:
  - all the servers must be in close physical proximity
  - there cannot be multiple, remote consoles
  - there is no way to get alerts from systems that need attention

Simple Network Management Protocol

- SNMP uses the network to report status information and alerts about remote systems
- SNMP messages are carried over UDP
- values can be loaded on demand (pull model), but when needed and configured appropriately, alerts are sent independently by the systems being managed (push)

SNMP
Management Information Base

- SNMP needs a machine-independent way to indicate which item of information is being requested or sent
- logically, the entire universe of information that can be accessed is built into a large tree: the Management Information Base or MIB
- the tree is extensible so individuals and organization can add their own subtrees -- private MIBs
- the tree is universal and known to all

navigating the MIB

- the path through the tree is sufficient to indicate one specific item (corresponding to a variable in a programming language)
- the path through the tree can be indicated by a sequence of numbers, the number of left siblings of the path being taken
- for example, 0.2.7.5.14.1.7.0 is such an Object Identifier (OID)
- OIDs are useful for enumerating arrays of objects, e.g., network interfaces, routing table entries

SNMP programs

- a network management station is used by the system administrator to monitor multiple systems
- a management agent must run on every managed device, get the required information, and provide it on request

SNMP basic operation

- the network management station may send GET requests to get one or more objects from specifc agents
- the network management station may also send SET requests to modify one or more objects on specifc agents
- agents will send TRAP or INFORM alerts to network management stations that they have been configured to alert
- because it uses UDP, SNMP (like DNS) cannot assume that its operations will be successful