

## Services 2

---

### DNS

#### Domain Name Service (DNS)

All TCP/IP communication uses IP addresses, like 171.64.64.250

The Domain Name Service allows us to use human readable names like “cse.stanford.edu” to refer to an IP address.

DNS is implemented on top of basic TCP/IP.

#### DNS Flexibility

The DNS system is built from many DNS servers arranged in a hierarchy that talk to each other to lookup the IP addr for a name.

A server of some sort on the Internet typically has a fixed IP addr and there's a DNS entry that points to that addr.

The DNS system gives us a level of indirection that allows us to replace or move a machine to a different IP address. Suppose we move the “www.foo.com” web site or email server from one machine to another and/or change its IP addr – so long as we update the DNS records for “www.foo.com”, everyone's requests will go to the new machine.

It's possible to do a reverse-DNS lookup – given an IP addr, find the DNS name.

It's possible for a DNS name to map to multiple IP addrs – machine can have multiple IP addrs ("multihoming"). Connecting to such a machine uses one of its IP addrs for the duration of the connection.

It's possible for multiple DNS names to all map to a single IP addr.

#### DNS Hierarchy

The DNS servers are arranged in a hierarchy.

Stanford's local DNS server knows the name→IP mapping for Stanford's machines – names ending in “.stanford.edu”

Somewhere in the world, when someone looks up the IP address for a Stanford address like “cslibrary.stanford.edu,” the request makes its way across the hierarchy to Stanford's DNS server.

However, there's a lot of caching of DNS addresses, so not every request actually traverses the whole hierarchy.

DNS entries are typically cached for 24 hours, so it can take 24 hours for a DNS change to propagate through the whole system.

#### DNS Hierarchy Traversal

Client program does a DNS lookup to get the IP addr of the DNS name. e.g. "www.yahoo.com" or "elaine23.stanford.edu".

Contacts the local DNS server. It may know the answer right away.

More likely, the local DNS server will pass the buck up to its parent DNS server.

The buck-passing follows a path among the DNS servers until it gets to a server that knows about that domain.

The answer tracks back to the local DNS server, which then (finally) answers the original DNS query.

### nslookup

On unix, the nslookup command can do DNS lookups or reverse-DNS lookups...

```
> nslookup www.stanford.edu
Server:  cicci.Stanford.EDU
Address: 171.64.7.121
```

```
Non-authoritative answer:
Name:    www.LB-A.stanford.edu
Address: 171.64.14.203
Aliases: www.stanford.edu
```

```
> nslookup www.yahoo.com
Server:  cicci.Stanford.EDU
Address: 171.64.7.121
```

```
Non-authoritative answer:
Name:    www.yahoo.akadns.net
Addresses: 66.218.71.84, 66.218.71.92, 66.218.71.90, 66.218.70.49
          66.218.71.93, 66.218.71.87, 66.218.71.94, 66.218.71.91, 66.218.70.48
          66.218.71.81, 66.218.71.95, 66.218.70.50, 66.218.71.88
Aliases: www.yahoo.com
```

```
> nslookup 171.64.14.203
Server:  cicci.Stanford.EDU
Address: 171.64.7.121
```

```
Name:    www1.Stanford.EDU
Address: 171.64.14.203
```

## **Email: SMTP, POP, IMAP**

### Anatomy of an email message send

Sender and recipient both have "accounts" on computers (source and destination hosts)

Compose message on source computer. Email is text based. Binary files such as GIF must be encoded to fit in an email message.

SMTP, Simple Mail Transfer Protocol, is used to transfer the email forward towards the destination.

On Unix, sending hands the message to a "sendmail" program on the source computer. If you're running a command line on the Unix host, you pass off to sendmail directly. If you are running on your PC, your mailer (Eudora, ...) will make and SMTP connection to your designated local SMTP server (e.g. smtp.stanford.edu), and it will handle the mail for you from there.

Source sendmail daemon tries to contact "SMTP" daemon on the destination host to send the message.

When contacted, the destination daemon accepts the email on behalf of the recipient. The destination daemon then copies the email into the user's "mailbox" where they will find it the next time they check.

Email "bouncing" back to sender account on error. It was an important for the social development of email that it feel "reliable" – so there's real effort to notify the sender if the message does not get through.

### Elm/Pine reading (old)

Mail files remain at your (Unix) email account.

You connect to your account for reading/sending via Telnet. No matter where you connect from, all of your old and new email is consistently there.

The leland system uses a slight variant of this where the email gets copied down into the users file space as soon as possible -- otherwise the load on the SMTP server to hold the incoming email would be quite high.

### POP reading (current)

Mail only temporarily on your email account host

Eudora/Netscape copies the mail down with the POP protocol to your client PC when you read it Handy GUI interface, but inconvenient to have your email copied down – you have to always read your email from the same place.

Standard POP sends the password in the clear – the APOP and KPOP variants avoid this problem.

### IMAP reading (future)

Mail remains on your email account host

The IMAP mail reader presents a GUI which allows you to see and manipulate your email conveniently wherever you are while the mail data remains on the server

The best of both worlds – this is essentially a thick-client client/server email reading solution.

## **Security / Authentication**

Most of our security coverage will come at the end of the quarter, but we'll do a little bit now to understand the security scheme in HW1

### Authentication

Prove who you are

3 ways

- Something you know – a password

- Something you are – fingerprint retinal scan (these methods have some serious disadvantages)

- Something you have – a little gadget which knows the password for you

## Traditional Password Auth

### "Shared Secret"

The password is a secret thing known to the client and server (or whoever the parties are).

Knowing the shared secret defines authenticity – only Bob knows the secret. Therefore, if X claims to be Bob, and X knows the secret, then X is Bob.

### Password demand

The server asks the client for the password. The client provides it, the server checks it against the server's copy of the password.

## One-Way Hash Function

### Hash

Given a string as input, combine/hash the bits together in a regular way to produce a "hash" value.

### Function of input

The hash value depends on every bit in the input. Changing the input a little results in a different hash value.

### Not-invertible

Given the hash value, it should not be possible (or very very hard) to compute the "inverse" – figure out what the original input was to result in that hash.

MD5 is a standard hash function.

## Hash password DB trick

How can you test for a thing, even if you don't know the exact value of that thing?

It's possible that the server does not need to store the actual password. Instead, the server will store a hash of the password.

When the client sends over a password, the server hashes it, and compares the hash value to the stored value to see if they match.

### Advantages

The server cannot impersonate the client.

Someone who breaks in to the server and steals the password db cannot impersonate the client.

### Disadvantages

The neat challenge/response schemes depend on the server knowing the actual password, so this trick is growing limited.

## Problem – passwords in the clear

If the client sends the password to the server just as plain text, then a bad guy who was eavesdropping could pick up the password, and later impersonate the client.

## Solution – challenge/response

### Challenge

The server sends a "challenge" to the client: a random number R.

### Client

The client computes  $\text{hash}(R + \text{password})$  and sends the hash to the server

### Verify

The server knows R and knows the password, so the server can also compute  $\text{hash}(R + \text{password})$  and see if it matches what the client sent.

Safe

Note that the bad guy can intercept R and the value of hash(R+password), but in theory cannot invert the hash function to find the original password.

### Problem Bad Passwords

Most ordinary people choose low quality passwords – obvious names, words, dates, ...

In fact, mostly bad passwords aren't a problem, since most online activities are not important – there's a lack of motive.

A motivated case: a bad guy breaks into a person's ebay account, and uses their good standing to do auction fraud.

In an increasingly electronic world, this will be more of a problem.

### Problem – password guessing

Bad guy

Knows R from observing the challenge

Knows hash(R+password) from observing the response

Cannot invert hash(R+password)

Can guess all possible passwords

Guess all possible passwords, and try hash(R+guess) for each one until a match is found to password(R+password)

Somewhat feasible

Say there are around 50,000 words ( $5e4$ ) in the English language

Combinations of two words =  $5e4 * 5e4 = 2.5 e9 = 2.5$  billion combinations

A large number, but not infeasible

If each letter may be upper/lower case, could increase the space by, say, a factor of 1000

Put the odds on your side by inserting letters, digits, punctuation randomly for important passwords. Don't use plain words, but slightly messed up words are ok.

Future problem

Password guessing will be a larger problem as computers get faster while the human capacity to remember random strings is fixed (or diminishing in my case!).

Possible Temporary Solution

Even though the server can compute the hash and compare very quickly, it can wait a second, or two, to respond to the client machine. That way, if the client was a bad guy, and trying to guess all 2.5 billion passwords, it would take at least 2.5 billion seconds or about 30,000 days to do it!

Future gadget

Eventually, we will need our ring/watch/pendant to do our authentication for us. This will make logins etc. more convenient – you walk up to the computer, and it just knows it's you.

### Encrypted Connection

Another solution would be to bring up the connection between the client and server so that all the traffic on it is encrypted. Then the bad guy cannot intercept anything, and we could go back to just sending the password.

This still suffers from the “man in the middle” attack – we'll study all these issues in detail later.