Types of Servers

Connectionless (UDP) verse Connection-Oriented (TCP)

Iterative verses Concurrent

Stateless verse stateful
Iterative

Single process

Handles requests one at a time

Good for low volume & requests that are answered quickly
Concurrent

Handle multiple requests concurrently

Normally uses thread/processes

Needed for high volume & complex requests

Harder to implement than iterative

Must deal with currency
State information

Information maintained by server about ongoing interactions with clients

State information cause problems

Consumes resources

How long does one maintain the state?
Stateless verses Stateful Servers

Stateless server

   Server that does not maintain state information

Stateful server

   Server that does maintain state information
HTTP & Server State

HTTP is stateless protocol

But need state for shopping carts etc.

Use Cookies to save state on client site

Privacy issues
Security issues
Stateless Protocols are easier

So students often transform stateful protocol into stateless protocol

Use cookie idea

Replay requests each time
Modes of Operation

Stateful servers sometimes have different modes of operation

Each mode has a set of legal commands

In Login mode only the commands password & username are acceptable

After successful login client-server connection in transaction mode

In transaction mode command X, Y Z are legal

These modes are also called server states or just states
Some Security
Places to attack

User

Client

Network

Server
User Attacks

Users select passwords that are easy to guess

Just ask the user for their password
Network attacks

Sniff network traffic

When user logs on view their password

telnet
HTTP
e tc.
Basic Http Authentication

To view this page, you need to log in to area “Access for /Campers” on www.scandiacampmendocino.org:80.
Your password will be sent in the clear.

Name: [ ]
Password: [ ]

[ ] Remember this password in my keychain

[ ] Cancel [ ] Log In
Requesting password protected page

Client Request
GET /private/index.html HTTP/1.0
Host: localhost

Server Response
HTTP/1.0 401 Authorization Required
Server: HTTPd/1.0
Date: Sat, 27 Nov 2004 10:18:15 GMT
WWW-Authenticate: Basic realm="Secure Area"
Content-Type: text/html
Content-Length: 311

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<html>
<head>
  <title>Error</title>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
</head>
<body><h1>401 Unauthorised.</h1></body>
</html>
User enters name and password

User enters (name "Aladdin", password "open sesame")

Browser sends
GET /private/index.html HTTP/1.0
Host: localhost
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

Server response:

HTTP/1.0 200 OK
Server: HTTPd/1.0
Date: Sat, 27 Nov 2004 10:19:07 GMT
Content-Type: text/html
Content-Length: 10476
Base64 Encoding

Encodes any byte sequence into sequence of printable characters

Encoded sequence can be decoded

Used to encode MIME contents for transport
  Email Attachments
### Base 64 Algorithm

Divide input into parts each part 24 bits long (3 bytes)

Convert each 24 bit sequence as follows:

Divide the 24 bits into four groups of 6 bits

Use the table to convert each 6 bits

<table>
<thead>
<tr>
<th>Value</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>25</td>
<td>Z</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>a</td>
</tr>
<tr>
<td>27</td>
<td>b</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>51</td>
<td>z</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>0</td>
</tr>
<tr>
<td>53</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>61</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>+</td>
</tr>
<tr>
<td>63</td>
<td>/</td>
</tr>
</tbody>
</table>

pad with =
## Example

<table>
<thead>
<tr>
<th>cats</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>001111111</th>
<th>00111101</th>
<th>01001010</th>
<th>01001001</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>001111</td>
<td>111001</td>
<td>111010</td>
<td>100101</td>
<td>001001</td>
</tr>
<tr>
<td>001111</td>
<td>111001</td>
<td>111010</td>
<td>100101</td>
<td>001001</td>
</tr>
<tr>
<td>15</td>
<td>57</td>
<td>58</td>
<td>37</td>
<td>9</td>
</tr>
<tr>
<td>p</td>
<td>5</td>
<td>6</td>
<td>l</td>
<td>j</td>
</tr>
</tbody>
</table>
Base64 Encoding & HTTP Authentication

Use Base64 encoding for user name and password

user name "Aladdin"
password "open sesame"

Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

Aladdin:open sesame

QWxhZGRpbjpvcGVuIHNlc2FtZQ==

Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
Base64 Decoding

Base 64 is designed to be decoded

Just reverse steps

So HTTP Authentication is not secure
  Same as sending user name and password as plain text
How to send passwords over network?

Use secure connection
SSL, TSL

Use one-way hash
One-Way Hash Functions

Let M be a message (sequence of bytes)

A one-way hash function f() such that:

- f maps arrays of bytes to arrays of bytes
- f(M) is always the same length
- Given an M it is easy to compute f(M)
- Given f(M) it is very hard/impossible to compute M
- Given M it is very hard/impossible to find N such that f(M) = f(N)

MD5 - Message Digest 5
SHA - Secure Hash Algorithm
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class OneWay
{
    public static void main(String args[])
    throws NoSuchAlgorithmException
    {
        MessageDigest sha = MessageDigest.getInstance("SHA");
        sha.update("Hi mom".getBytes());
        byte[] shaHash = sha.digest();
        System.out.println(new String(shaHash));

        MessageDigest md5 = MessageDigest.getInstance("MD5");
        md5.update("Hi mom".getBytes());
        byte[] md5Hash = md5.digest();
        System.out.println(new String(md5Hash));
    }
}
Hex Representation

Usually one converts sha/md5 hash to
Base 64
Hex

```java
static final String HEXES = "0123456789ABCDEF";
public static String getHex( byte[] raw ) {
  if ( raw == null ) {
    return null;
  }
  final StringBuilder hex = new StringBuilder( 2 * raw.length );
  for ( final byte b : raw ) {
    hex.append(HEXES.charAt((b & 0xF0) >> 4))
        .append(HEXES.charAt((b & 0x0F)));
  }
  return hex.toString();
}
```

lost the reference to the source of this code, but it is fairly common
Using one-way hash to send password

Client
- Requests nonce from server
- Client computes hash(password + nonce)
- Client sends hash(password + nonce) & nonce back to server

Server
- Gets hash(password + nonce) & nonce
- Reads password from file
- Computes hash(password + nonce)
- Compares value with one client sent

nonce
- String that is used only once
- Should be longer than 48 bits
What the attacher sees

nonce
hash(password + nonce)

but hash is one way so can not reverse it
How they can break this system

They know

nonce
hash(password + nonce)

Compute table containing

word  hash(word + nonce)

Do it for all

words in dictionary
List of potential passwords

Now do reverse look up on hash(password + nonce)
How to defeat look up trick

Use good password
   multiple words
   Mix cases
   Use numbers and other characters

Use Key stretching
Key Stretching

Compute hash more than once

key = ""
for 1 to 65536 do
    key = hash(key + password + nonce)

Then client sends key

This means it will take a lot longer for attacher to build table

reference http://en.wikipedia.org/wiki/Key_stretching
Password Files

If password files contains password then attacher just breaks into server and gets all the passwords
Salting the Password File

Password File

<table>
<thead>
<tr>
<th>name</th>
<th>hash</th>
<th>salt</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo</td>
<td>hash(password1+salt1)</td>
<td>salt1</td>
</tr>
<tr>
<td>bar</td>
<td>hash(password2+salt2)</td>
<td>salt2</td>
</tr>
</tbody>
</table>

Client sends server password over secure connection

Server validates buy computing hash(password+salt)