Lecture 25– Web Security

Stephen Checkoway
Oberlin College
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 our 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

Crash course on HTTP

- Hypertext transfer protocol
- Standard protocol to access documents on the web
- Multiple versions 1.0, 1.1, 2, and 3; web servers that support version n usually support older versions too
- HTTP 1.0 and 1.1 are text protocols

URLs

- URLs give names to resources
- Examples:
  - http://example.com
  - https://subdomain.example.org/file/path
  - https://example.com/blah?key1=value1&key2=value2
  - https://example.com/#fragment
- URL consists of
  - A scheme (http or https) followed by ://
  - A domain (example.com or subdomain.example.org)
  - Optionally a path to a resource (/file/path)
  - Optionally a query string which is generally a key-value pairs separated by &
  - Optionally a fragment which names a portion of the resource

HTTP Request

- Browser makes a request by sending a request line followed by request headers followed by a blank line (lines end with \r\n), optionally followed by a body
- Example request line
  - GET /path/to/file.html HTTP/1.1
  - GET /index.html?var1=value1&var2=value2 HTTP/1.1
  - PUT /path/to/file.html HTTP/1.1

HTTP methods

- HTTP defines multiple methods, the most common are GET and PUT
- GET requests are used for accessing server data with request data encoded in the request line
  - Two identical GET requests should return identical results (unless server state has been updated between the request)
- POST requests are used for changing state on the server (like logins or sending data to the server); request data is encoded in the HTTP request body rather than the request line

HTTP responses

- Upon receiving a HTTP request, a server responds with a response
- Response consists of a response line containing a status code, headers, a blank line, and then the body of the accessed resource

Successful request & response

$ nc -c example.com 80
GET / HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
Date: Mon, 06 Apr 2026 19:23:47 GMT
Content-Type: text/html
…

<!doctype html><html lang="en"><head><title>Example Domain</title><meta name="viewport" content="width=device-width, initial-scale=1"><style>body{background:#eee;width:60vw;margin:15vh auto;font-family:system-ui,sans-serif}h1{font-size:1.5em}div{opacity:0.8}a:link,a:visited{color:#348}</style></head><body> <h1>Example Domain</h1> This domain is for use in documentation examples without needing permission. Avoid use in operations. Learn more </body></html>

Code Injection

```
<?php

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

GET /?path=/home/user/ HTTP/1.1

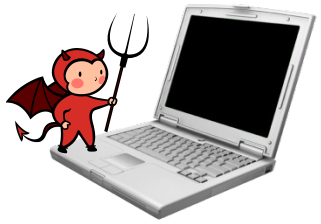
HTTP/1.1 200 OK
…
Desktop
Documents
Music
Pictures

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 Oberlin?

```
SELECT COUNT(*) FROM `users` WHERE `location` = ‘Oberlin’
```

- 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 = “Oberlin’; DELETE FROM `users` WHERE 1=‘1”

```
SELECT * FROM `users` WHERE `location`=‘Oberlin’; DELETE FROM `users` WHERE 1=‘1’
```

This is interpreted as:

```
SELECT * FROM `users` WHERE `location` = ‘Oberlin’; DELETE FROM `users` WHERE 1=‘1’
```

SQL Injection

- Attacker embedded **code** in the form of SQL statements inside **data**
- The website interpolated the attacker's string inside the intended SQL query and passed it along to the database
- The database saw a valid query—in this case, containing two statements—and executed it

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
```

Bad approach: escaping characters

- Characters that can change how the query is interpreted get escaped
- PHP example: mysqli::real_escape_string(string_from_user)
  - The string_from_user will insert a backslash before NULL (ASCII 0), newlines (\n), carriage returns (\r), commas, single quotes, and double quotes
- This only works if the attacker-controlled string is actually used inside quotes.

Good approach: Prepared statements

- Separates the query from the data that the query operates on
- Different databases use different syntaxes but the concrete data is replaced with a wildcard like ? to produce a prepared statement

```
$pstmt = $db->prepare("SELECT * FROM `users` WHERE location=?");
```

- A separate database statement is used to execute the query along with the arguments to the query

```
$pstmt->execute(array($city));
```

Blind SQL injection

- Often the website's response to SQL injection will immediately show if the injection was successful or not by displaying the output of the database (or logging the attacker in)
- Sometimes the website's response doesn't do this but can still be vulnerable to blind SQL injections
- Attacker injects a sequence of true/false queries, each of which tries to learn one bit of information
  - If the attacker can distinguish a true from a false, then they can learn the data
  - Conditionally causing the database to take different amounts of time can work when the website's response is always the same

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”

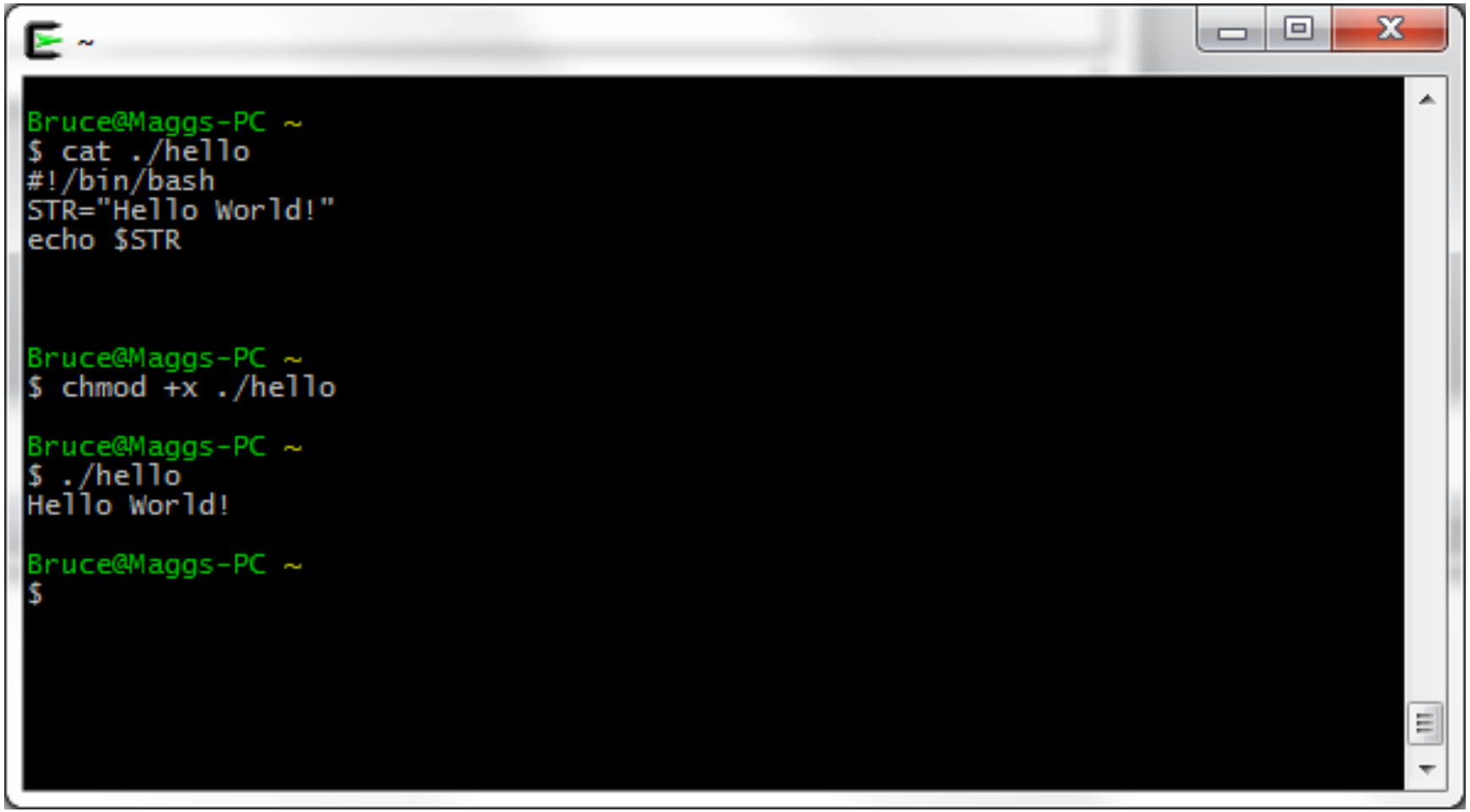
#!/path/to/program

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 (often 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)

Perl script that prints 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

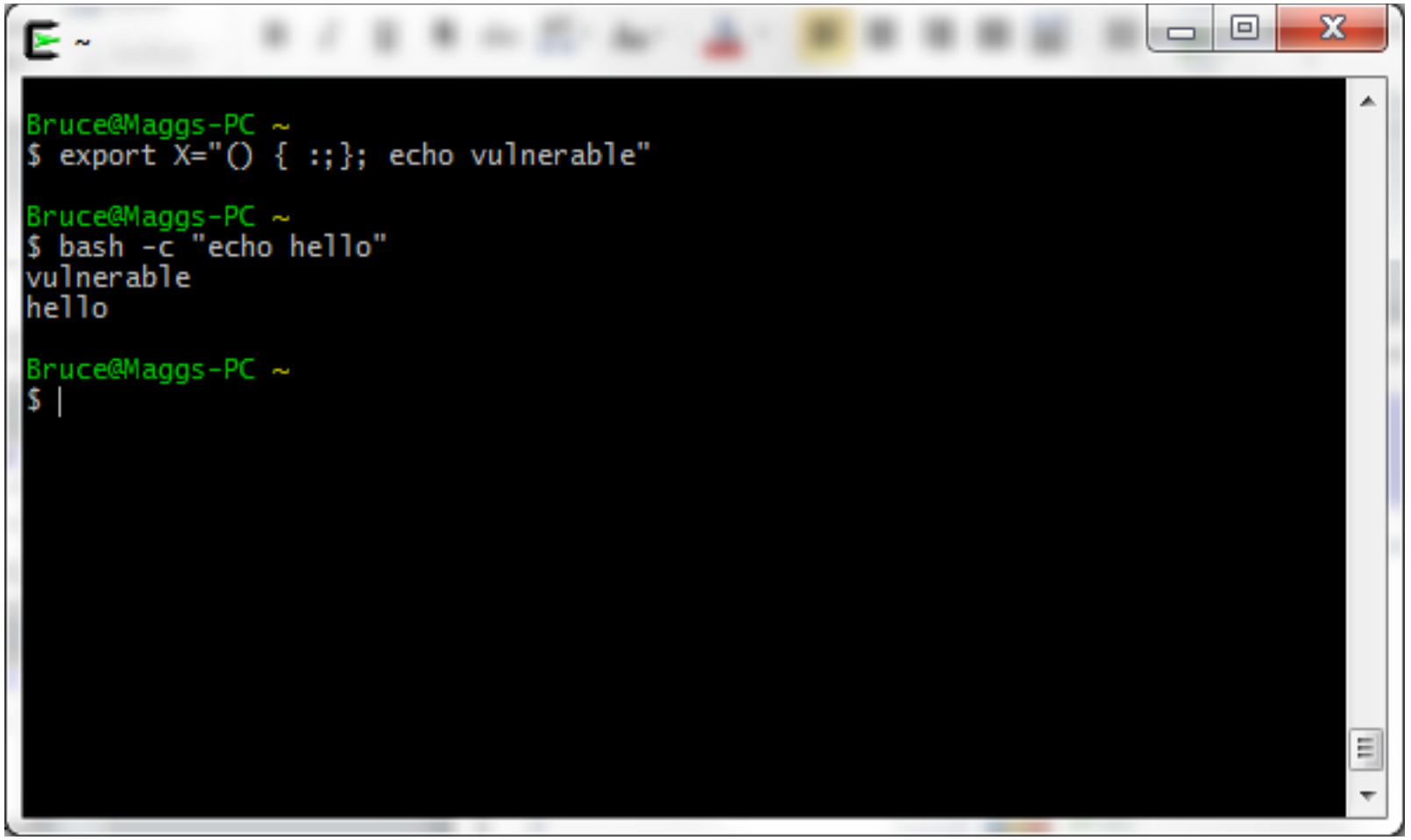
```
DOCUMENT_ROOT="C:/Program Files (x86)/Apache Software
Foundation/Apache2.2/htdocs“
GATEWAY_INTERFACE="CGI/1.1“
HOME="/home/SYSTEM"
HTTP_ACCEPT="text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8“
HTTP_ACCEPT_CHARSET="ISO-8859-1,utf-8;q=0.7,*;q=0.7"
HTTP_ACCEPT_ENCODING="gzip, deflate“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

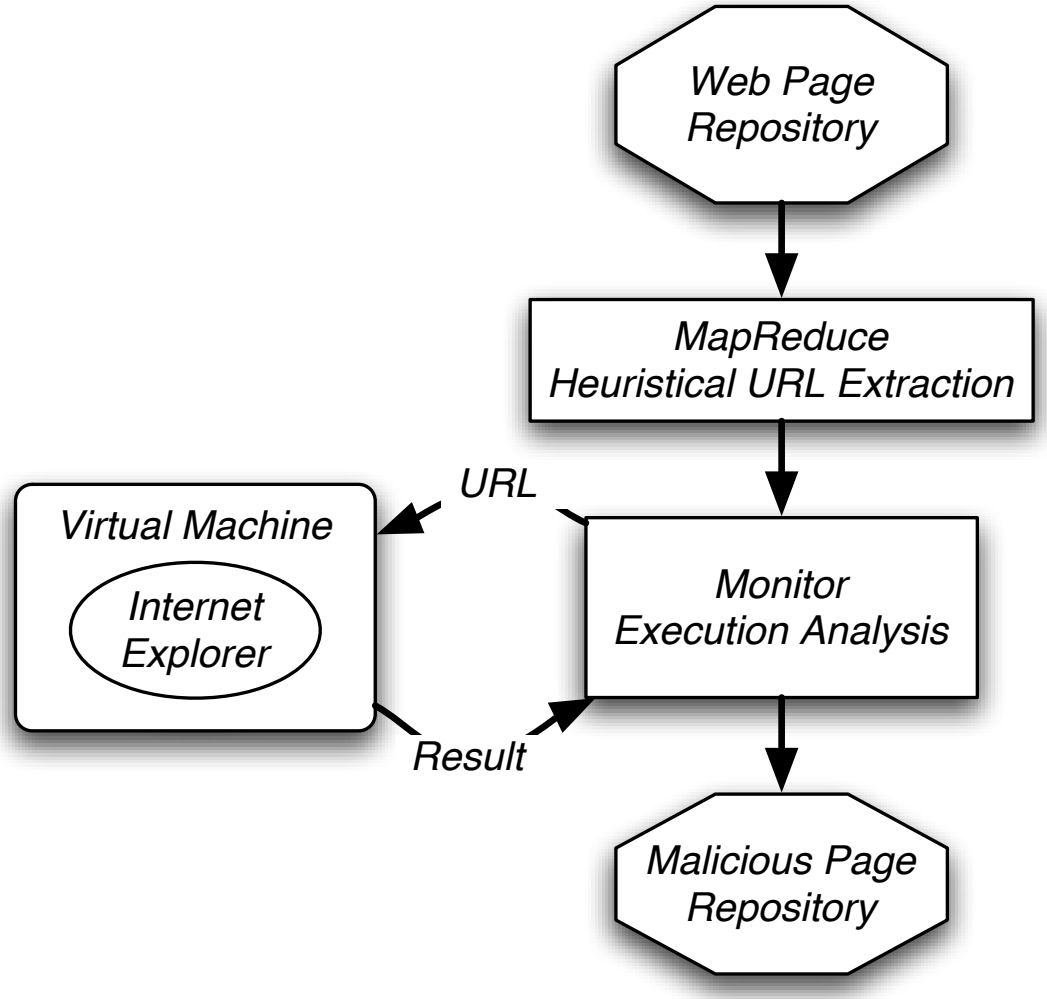
Introduction

- 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

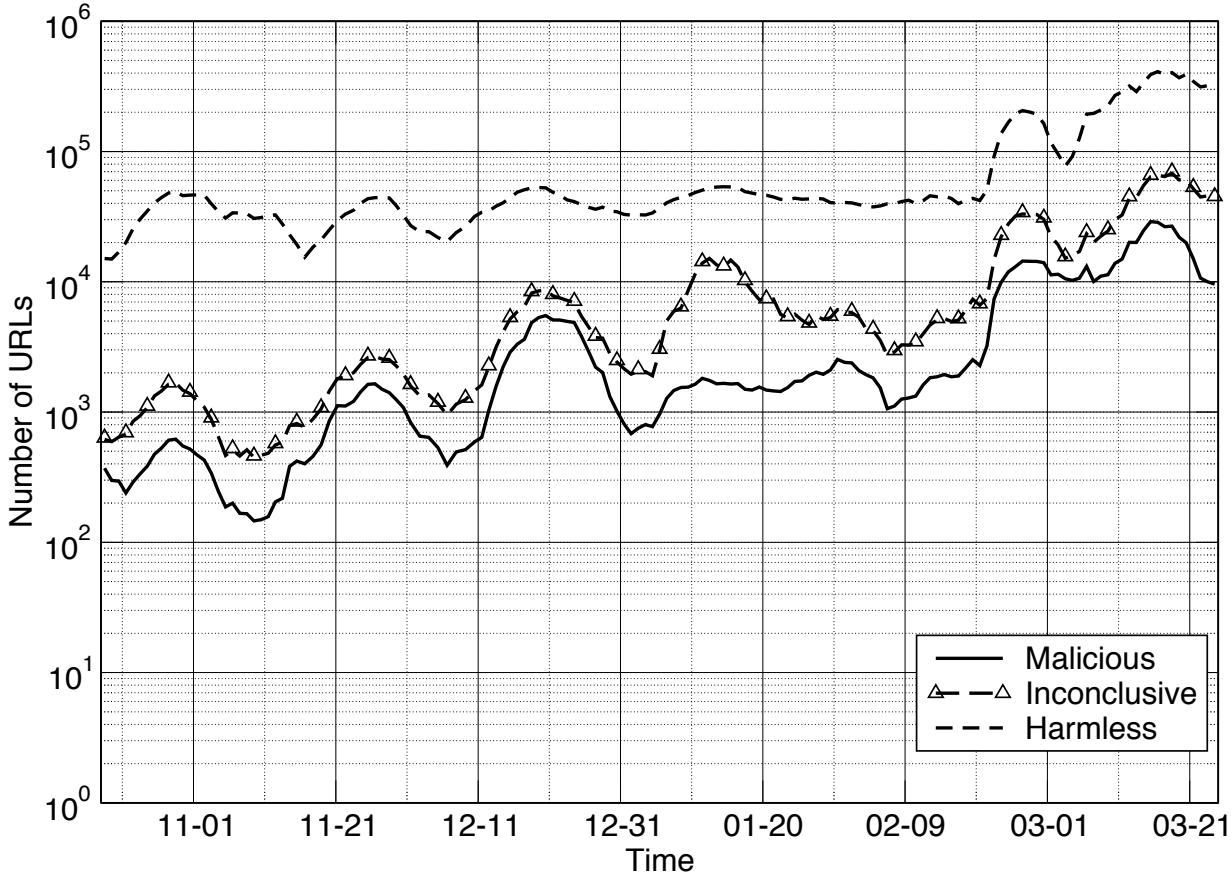
- Malicious website automatically installs malware on visitor's computer
  - usually via exploits in the browser or other software on the client (without user consent)
- Authors use Google's infrastructure to analyze several billion URLs

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

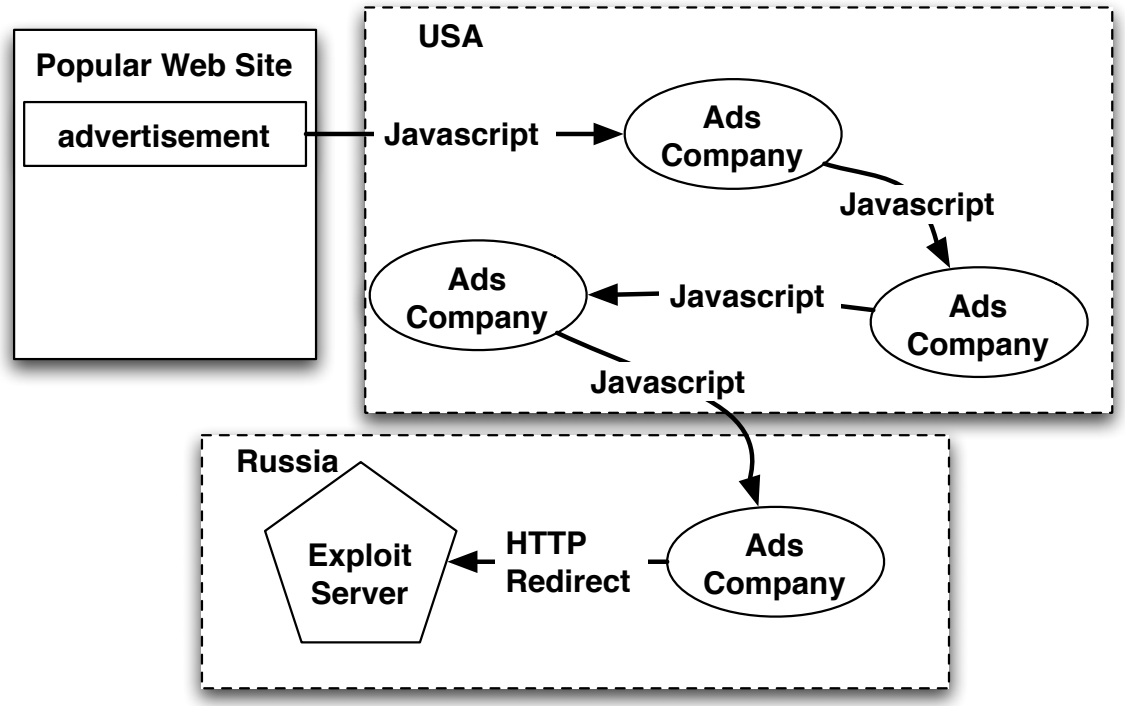
Web Server Security

- 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 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 the video, 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