Web Security

1

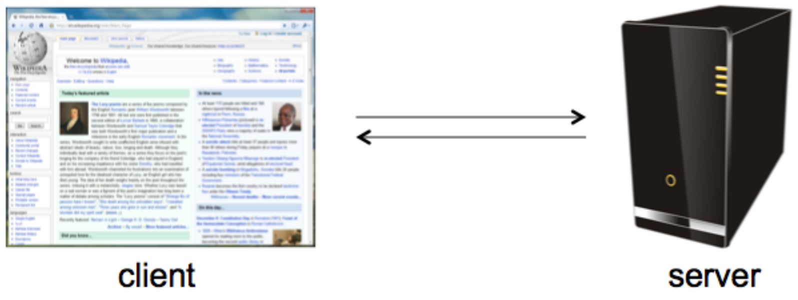
Today

- Web architecture
  - Basics of web security
- Shellshock vulnerability
- Introduction to HTTPS

2

What is the Web?

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



3

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

4

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

5

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

6

HTTP protocol

- HTTP is
  - widely used
  - Simple
  - Stateless

7

URLs

- Global identifiers of network-retrievable documents
- Example:

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

Protocol

Path

Query

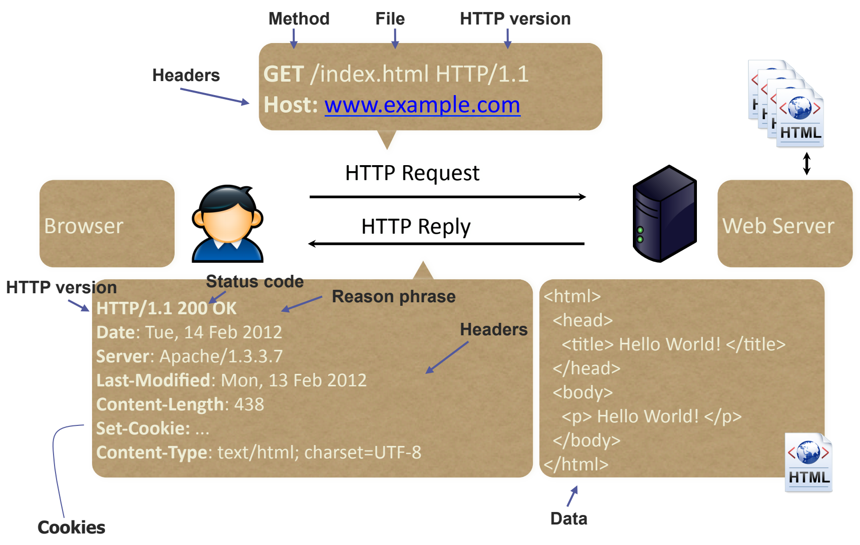
Fragment

What is the difference between URL and URI?

Are URLs case-sensitive?

8

HTTP Protocol



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

```
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>
```

9

Security on the web

- Integrity
  - malicious web sites should not be able to tamper with integrity of my computer or my information on other web sites
- Confidentiality
  - malicious web sites should not be able to learn confidential information from my computer or other web sites
- Privacy
  - malicious web sites should not be able to spy on me or my activities online

10

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.

11

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

12

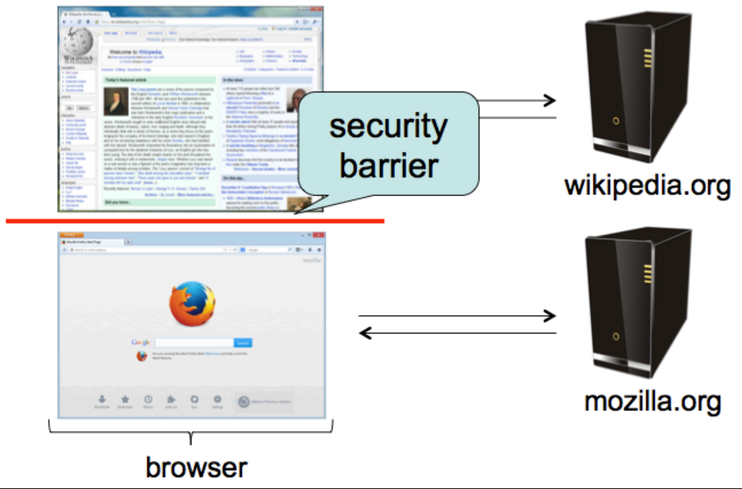
Security on the web

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

13

Same-origin policy

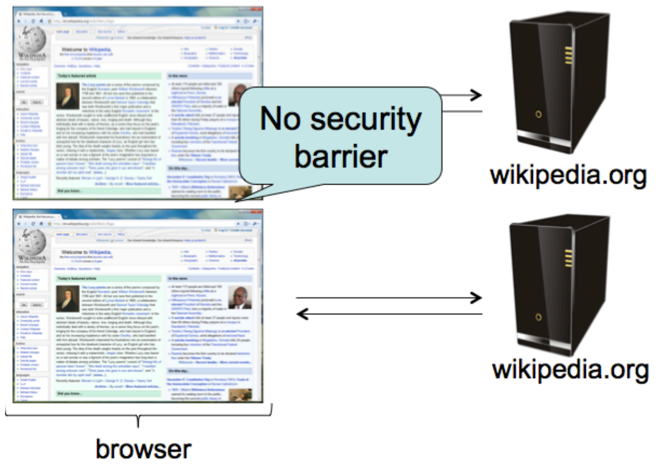
- Each site is isolated from all others



14

Same-origin policy

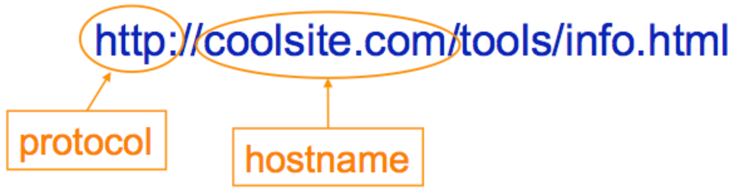
- Multiple pages from same site aren't isolated



15

Same-origin policy

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



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

16

Same-origin policy

- The origin of a page (frame, image, ...) is derived from the URL it was loaded from
- Special case: Javascript runs with the origin of the page that loaded it

17

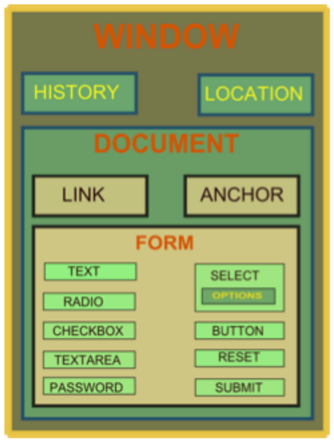
Dynamic Web Pages

- 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>
```

18

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."

19

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!)*

20

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>
```

21

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!

22

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**

23

Cookies

- Cookies are a small bit of information stored on a computer associated with a specific server
  - When you access a specific website, it might store information as a cookie
  - Every time you revisit that server, the cookie is re-sent to the server
  - Effectively used to hold state information over sessions
- Cookies can hold any type of information
  - Can also hold sensitive information
    - This includes passwords, credit card information, social security number, etc.
    - Session cookies, non-persistent cookies, persistent cookies
  - Almost every large website uses cookies

24

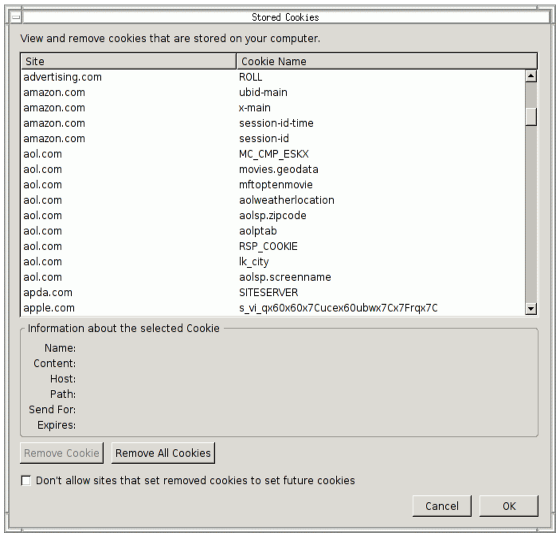
More on Cookies

- Cookies are stored on your computer and can be controlled
  - However, many sites require that you enable cookies in order to use the site
  - Their storage on your computer naturally lends itself to exploits (Think about how ActiveX could exploit cookies...)
  - You can (and probably should) clear your cookies on a regular basis
  - Most browsers will also have ways to turn off cookies, exclude certain sites from adding cookies, and accept only certain sites' cookies
- Cookies expire
  - The expiration is set by the sites' session by default, which is chosen by the server
  - This means that cookies will probably stick around for a while

25

Taking Care of Your Cookies

- Managing your cookies in Firefox:
  - Remove Cookie
  - Remove All Cookies
  - Displays information of individual cookies
  - Also tells names of cookies, which probably gives a good idea of what the cookie stores
    - i.e. amazon.com: session-id



View and remove cookies that are stored on your computer.

Site	Cookie Name
advertising.com	ROLL
amazon.com	ubid-main
amazon.com	x-main
amazon.com	session-id-time
amazon.com	session-id
aol.com	MC_CMP_ESKX
aol.com	movies.geodata
aol.com	mftoptenmovie
aol.com	aolweatherlocation
aol.com	aolsp.zipcode
aol.com	aolptab
aol.com	RSP_COOKIE
aol.com	lk_city
aol.com	aolsp.screenname
apda.com	SITESERVER
apple.com	s_vi_qx60x60x7Cucex60ubwx7Cx7Frqx7C

Information about the selected Cookie

Name:
Content:
Host:
Path:
Send For:
Expires:

☐ Don't allow sites that set removed cookies to set future cookies

26

Exercise

- Ad servers are increasingly being used to display essential content for web sites. Suppose that the same host is used to serve images for two different web sites.
  1. Explain why this is a threat to user privacy.
  2. Is this threat eliminated if the browser is configured to reject third-party cookies?

27

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

Acknowledgement: slides from Prof. Bruce Maggs

28

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

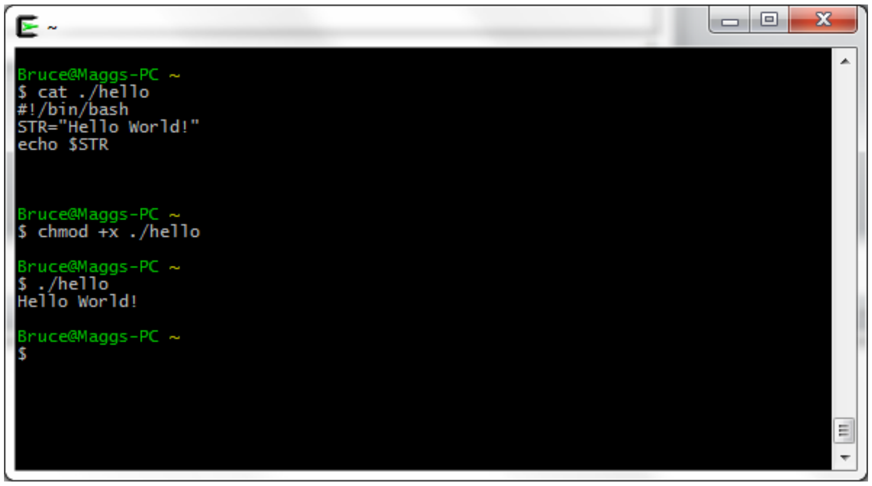
Tells bash to pass the rest of the file to program to be executed.

Example:

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

31

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 ~
$
```

32

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

35

CGI Output

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

36

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`

37

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/system32:...“
PATH_INFO="/foo/bar“
QUERY_STRING="var1=value1&var2=with%20percent%20encoding
```

38

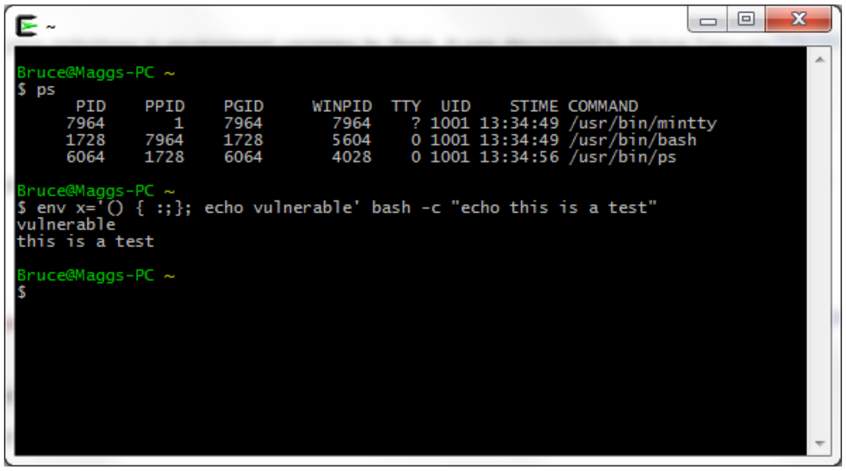
Shellshock Vulnerability

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

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

39

Cygwin Bash Shell Shows Vulnerability

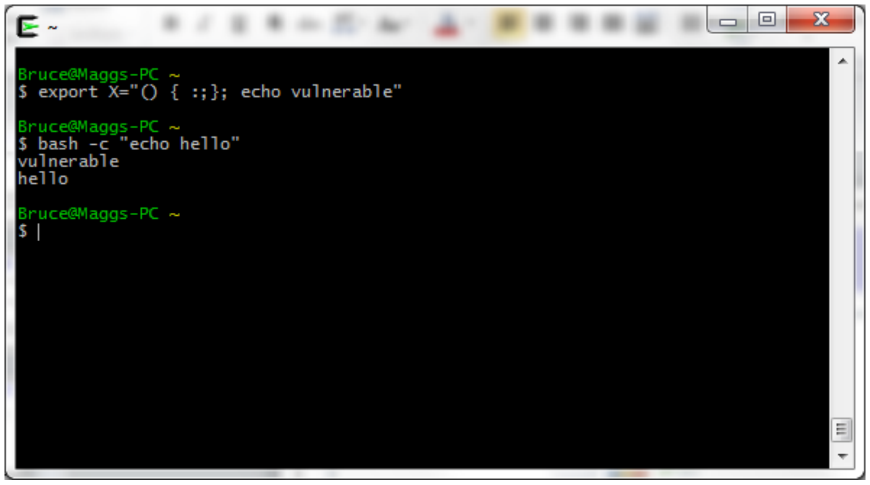


```
Bruce@Maggs-PC ~
$ ps
      PID    PPID    PGID     WINPID  TTY  UID    STIME COMMAND
     7964       1    7964       7964    ? 1001 13:34:49 /usr/bin/mintty
     1728    7964    1728       5604    0 1001 13:34:49 /usr/bin/bash
     6064    1728    6064       4028    0 1001 13:34:56 /usr/bin/ps

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!

40

Alternatively



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

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

Bruce@Maggs-PC ~
$ |
```

41

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.

42

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

43

Purported WopBot Attack on Akamai

There have been news reports indicating that Akamai was a target of a recent ShellShock-related BotNet attack. (See information about WopBot). Akamai did observe DDOS commands being sent to a IRC-controlled botnet to attack us, although the scale of the attack was insufficient to trigger an incident or need for remediation. Akamai was not compromised, nor were its customers inconvenienced. We receive numerous attacks on a daily basis with little or no impact to our customers or the services we.

https://blogs.akamai.com/security/

44

HTTPS

45

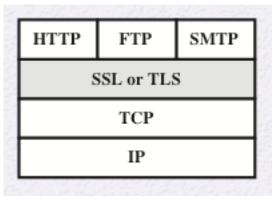
SSL

- **SSL** (Secure Socket Layer) -- Netscape
  - Version 2.0 -- Broken, don't use (disabled by default in modern browsers)
  - Version 3.0 (older but still in use)
- **TLS** (Transport Layer Security) -- IETF Standard
  - Version 1.0, 1.1, 1.2 (commonly used),
  - Version 1.3 (draft)

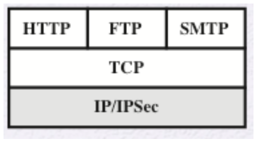
46

Different locations of security implementations

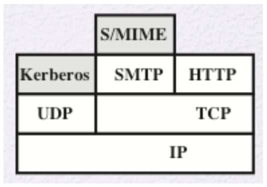
- Transport level



- Network level



- Application level



47

Threat model

- Controls infrastructure (routers, DNS, wireless access points)
- Passive attacker: only eavesdrops
- Active attacker: eavesdrops, injects, blocks, and modifies packets
- What examples?
  - Internet Cafe, hotel, CSE, fake web site,
- Does not protect against:
  - Intruder on server
  - Spyware on client
  - SQL injection, XSS, CSRF

48

Public-key cryptography

- Bob generates: SK_Bob, PK_Bob
- Alice can encrypt messages to Bob:
  - using PK_Bob encrypts messages, only Bob can decrypt
- Bob can sign messages that Alice can verify:
  - using SK_Bob signs message, anyone can verify
- What was the hard problem we haven't yet solved about this?

49

Certificates

- This public key with SHA-256 hash (XXX) belongs to the site (name, e.g., Amazon.com)
  - Signed by a trusted authority (digital signature)
- Your browsers (e.g., Firefox, Chrome) trust a specific set of CAs as root CAs
  - Shipped with the public keys of the root CAs

50

Certificates

How does Alice (web browser) obtain PK_Bob?

Browser (Alice)	**Server (Bob)**	**Certificate Authority (CA)**
		[Think of like a notary]
(Knows PK_CA)		(Keeps SK_CA Secret)
	1. Choose (SK,PK)	
	--- PK and proof he is "Bob" -->	
		2. Checks proof
	<-- Signs certificate with SK_CA ---- "Bob's key is PK -- Signed, CA"	
	3. Keeps cert on file	

<-- Sends cert to Alice ----
"Bob's key is PK -- Signed, CA"

4. Verifies signature on cert.

51

Certificates verification

- How is identity verification done?
  - Typically 'DV' (domain verification) – just an email based challenge to the address in the domain registration records (Or some default email address); minimally secure.
- Cert has expiration date (e.g., one year ahead) [-- Why?]

52

SSL Certificates

- A trusted authority vouches that a certain public key belongs to a particular site
- Format called x.509 (complicated)
- Browsers ship with CA public keys for large number of trusted CAs [accreditation process]
- Important fields:
  - Common Name (CN) [e.g., *.google.com]
    Expiration Date [e.g. 2 years from now]
    Subject's Public Key
    Issuer -- e.g., Verisign
    Issuer's signature
- Common Name field
  - Explicit name, e.g. eecs.umich.edu
  - Or wildcard, e.g. *.umich.edu

53

Summary

- Today's lecture
  – Web architecture
  – Shellshock vulnerability
- Wed's lecture
  – HTTPS, Part 2

54