Web Security
Computer Security
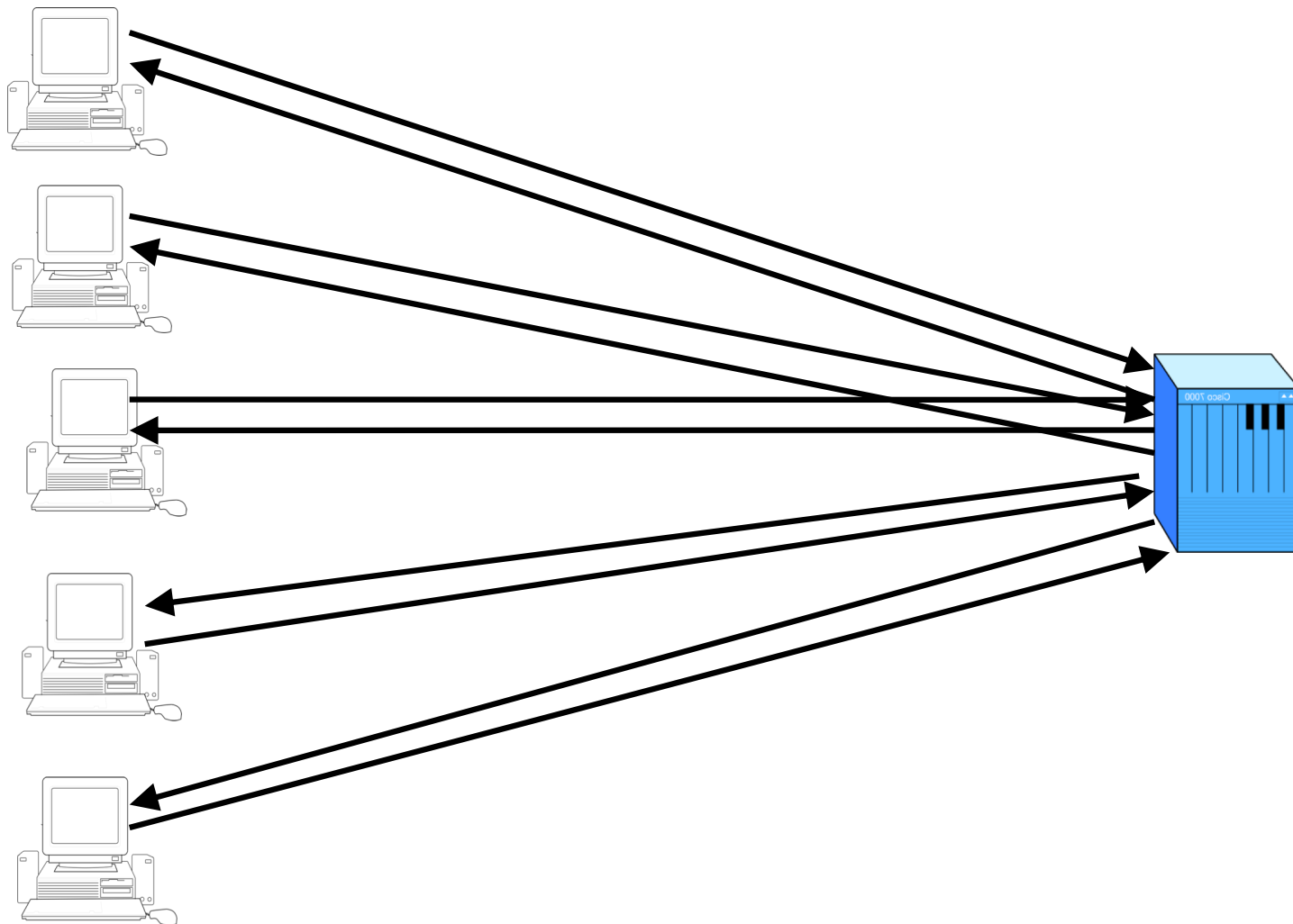# Peter Reiher
February 25, 2021

Web Security

- Lots of Internet traffic is related to the web

- Much of it is financial in nature

- Also lots of private information flow around web applications

- An obvious target for attackers
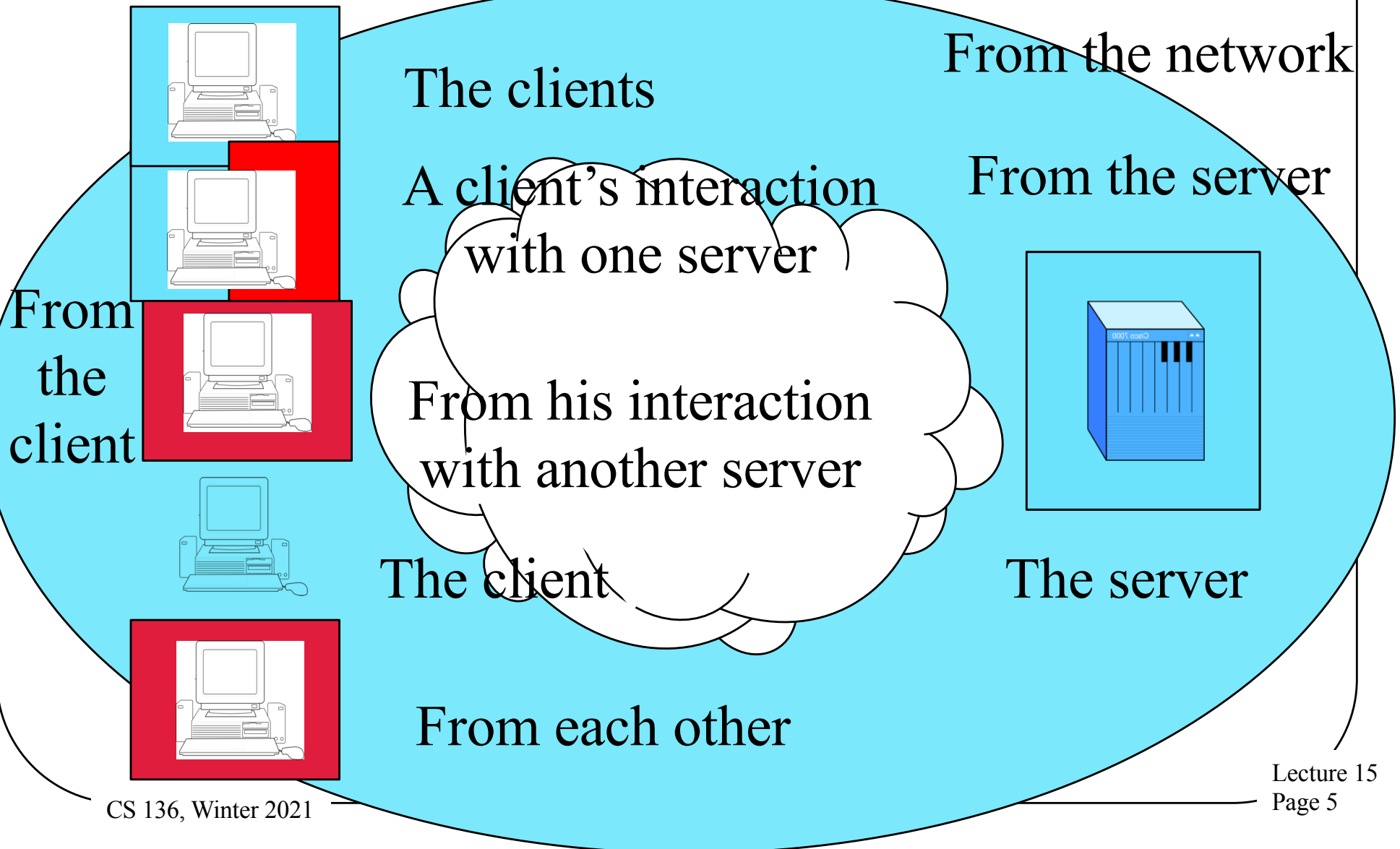
The Web Security Problem

- Many users interact with many servers

- Most parties have little other relationship

- Increasingly complex things are moved via the web

- No central authority

- Many developers with little security experience

- Many critical elements originally designed with no thought to security

- Sort of a microcosm of the overall security problem

Aspects of the Web Problem

Who Are We Protecting?

Everyone

The clients

From the network

A client's interaction with one server

From the server

From the client

From his interaction with another server

The client

The server

From each other

What Are We Protecting?

- The client's private data

- The server's private data

- The integrity (sometimes also secrecy) of their transactions

- The client and server's machines

- Possibly server availability

  – For particular clients?

Some Real Threats

- Buffer overflows and other compromises
  - *Client attacks server*

- Web based social engineering attacks
  - *Client or server attacks client*

- SQL injection
  - *Client attacks server*

- Malicious downloaded code
  - *Server attacks client*

More Threats

- Cross-site scripting

  – *Clients attack each other*

- Threats based on non-transactional nature of communication

  – *Client attacks server*

- Denial of service attacks

  – *Threats on server availability (usually)*

Yet More Threats

- Browser security

  – Protecting interactions from one site from those with another

  – *One server attacks client's interactions with another*

- Data transport issues

  – *The network attacks everyone else*

- Certificates and trust issues

  – *Varied, but mostly server attacks client*

Compromise Threats

- Much the same as for any other network application

- Web server might have buffer overflow

  – Or other remotely usable flaw

- Not different in character from any other application's problem

  – And similar solutions

Solution Approaches

- Patching

- Use good code base

- Minimize code that the server executes

- Maybe restrict server access

  – When that makes sense

- Lots of testing and evaluation

  – Many tools for web server evaluation

Compromising the Browser

- Essentially, the browser is an operating system
  - You can do almost anything through a browser
  - It shares resources among different "processes"
- But it does not have most OS security features
- While having some of the more dangerous OS functionality
  - Like arbitrary extensibility
  - And supporting multiple simultaneous mutually untrusting pieces of code

But My Browser Must Be OK . . .

- After all, I see the little lock icon at the bottom of the page

- Doesn't that mean I'm safe?

- Alas, no

- What does that icon mean, and what is the security implication?

The Lock Icon

- This icon is displayed by your browser when some security measures are in place

- A web site provided a certificate attesting to its identity

  – And it's using HTTPS

- The certificate was properly signed by someone your browser trusts

- It doesn't necessarily mean the site is trustworthy

What Are the Implications?

- All you know is that the web site is who it claims to be
  - Which might not be who you think it is
  - Maybe it's `amazon.com`, not `amazon.com`
  - Would you notice the difference?
- Only to the extent that a trusted signer hasn't been careless or compromised
  - Some have been, in the past

Another Browser Security Issue

- What if you're accessing your bank account in one browser tab

- And a site showing silly videos of cats in another?

- What if one of those videos contains an attack script?

- Can the evil cat script steal your bank account number?

Same Origin Policy

- Meant to foil such attacks

- Built into all modern browsers

- Basically, pages from a single origin can access each other's stuff

- Pages from different origins cannot access each others' stuff

- Particularly relevant to cookies

Web Cookies

- Essentially, data that a web site asks your browser to store

- Sent back to that web site when you ask for another service from it

- Used to set up sessions and maintain state (e.g., authentication status)

- Lots of great information about your interactions with sites is in the cookies

Same Origin Policy and Cookies

- Script from one domain cannot get the cookies from another domain
  - Prevents the evil cat video from sending authenticated request to empty your bank account
- Domain is defined by DNS domain name, application protocol
  - Sometimes also port

A Cookie Caveat

- Modern web pages are actually composed of lots of pieces

  - Many not created by the entity you think you're talking to

- Those other entities can give you cookies, too

- And they can use them to track you

  - E.g., cookie from sneaky.com picked up on one web page can be delivered back to sneaky.com on another

SQL Injection Attacks

- Many web servers have backing databases

  – Much of their information stored in a database

- Web pages are built (in part) based on queries to a database

  – Possibly using some client input . . .

SQL Injection Mechanics

- Server plans to build a SQL query
- Needs some data from client to build it
  - E.g., client's user name
- Server asks client for data
- Client, instead, provides a SQL fragment
- Server inserts it into planned query
  - Leading to a "somewhat different" query

An Example

```
"select * from mysql.user
   where username = ' " . $uid . " ' and
   password=password(' ". $pwd " ');"
```

- Intent is that user fills in his ID and password

- What if he fills in something else?

```
'or 1=1; -- '
```

What Happens Then?

- $uid has the string substituted, yielding

```
"select * from mysql.user
    where username = ' ' or 1=1; -- ' ' and
    password=password(' ". $pwd " ');"
```

- This evaluates to true

  - Since 1 does indeed equal 1

  - And `--` comments out rest of line

- If script uses truth of statement to determine valid login, attacker has logged in

Basis of SQL Injection Problem

- ## **Unvalidated input**

- Server expected plain data

- Got back SQL commands

- Didn't recognize the difference and went ahead

- Resulting in arbitrary SQL query being sent to its database

  – With its privileges

Some Example Attacks

- 130 million credit card numbers stolen in 2009 with SQL injection attack

- Used to steal 1 million Sony passwords

- FortiWeb firewalls had one (2021)

- A WordPress theme had one (2020)

- vBulletin had one (2020)

- Sophos SG firewall had one (2020)
  – That was exploited

Solution Approaches

- Carefully examine all input

- Avoid using SQL in web interfaces

- Parameterized variables

Examining Input for SQL

- SQL is a well defined language

- Generally web input shouldn't be SQL

- So look for it and filter it out

- **Problem**: proliferation of different input codings makes filtering hard

- **Problem**: some SQL control characters are widely used in real data

  – E.g., apostrophe in names

Avoid SQL in Web Interfaces

- Never build a SQL query based on user input to web interface

- Instead, use predefined queries that users can't influence

- Typically wrapped by query-specific application code

- **Problem**: may complicate development

Use Parameterized Variables

- SQL allows you to set up code so variables are bound parameters

- Parameters of this kind aren't interpreted as SQL

- Pretty much solves the problem, and is probably the best solution

Malicious Downloaded Code

- The web relies heavily on downloaded code
  - Full language and scripting language
  - Mostly scripts
- Instructions downloaded from server to client
  - Run by client on his machine
  - Using his privileges
- Without defense, script could do anything

Types of Downloaded Code

- Java

  - Full programming language

- Scripting languages

  - JavaScript

  - VB Script

  - ECMAScript

  - XSLT

Drive-By Downloads

- Often, user must request that something be downloaded

- But not always

  – Sometimes visiting a page or moving a cursor causes downloads

- These are called *drive-by downloads*

  – Since the user is screwed just by visiting the page

Solution Approaches

- Disable scripts in your browser

- Use secure scripting languages

- Isolation mechanisms

- Virus protection and blacklist approaches

Disabling Scripts

- Browsers (or plug-ins) can disable scripts

  – Selectively, based on web site

- The bad script is thus not executed

- **Problem**: Cripples much good web functionality

  – So users re-enable scripting

Use Secure Scripting Languages

- Some scripting languages are less prone to problems than others

- Write your script in those

- **Problem**: secure ones aren't popular

- **Problem**: many bad things can still be done with "secure" languages

- **Problem**: can't force others to write their scripts in these languages

Isolation Mechanisms

- Architecturally arrange for all downloaded scripts to run in clean VM

  – Limiting the harm they can do

- **Problem**: they might be able to escape the VM

- **Problem**: what if a legitimate script needs to do something outside its VM?

Signatures and Blacklists

- Identify known bad scripts
- Develop signatures for them
- Put them on a blacklist and distribute it to others
- Before running downloaded script, automatically check blacklist
- **Problem**: same as for virus protection

Cross-Site Scripting

- XSS

- Many sites allow users to upload information
  - Blogs, photo sharing, Facebook, etc.
  - Which gets permanently stored
  - And displayed

- Attack based on uploading a script

- Other users inadvertently download it
  - And run it . . .

The Effect of XSS

- Arbitrary malicious script executes on user's machine

- In context of his web browser

  – At best, runs with privileges of the site storing the script

  – Often likely to run at full user privileges

Non-Persistent XSS

- Embed a small script in a link pointing to a legitimate web page

- Following the link causes part of it to be echoed back to the user's browser

- Where it gets executed as a script

- Never permanently stored at the server

Persistent XSS

- Upload of data to a web site that stores it permanently

- Generally in a database somewhere

- When other users request the associated web page,

- They get the bad script

Some Examples

- A WordPress plugin (2020)

- Other XSS vulnerabilities discovered on sites run by Symantec, PayPal, Facebook, LinkedIn, Adobe, Apple App Store, Google Gmail, Electron Application Framework, thousands of others

- D-Link router flaw exploitable through XSS

Why Is XSS Common?

- Use of scripting languages widespread
  - For legitimate purposes
- Most users leave them enabled in their browsers
- Sites allowing user upload are very popular
- Only a question of getting user to run your script

Typical Effects of XSS Attack

- Most commonly used to steal personal information

  - That is available to legit web site

  - User IDs, passwords, credit card numbers, etc.

- Such information often stored in cookies at client side

Solution Approaches

- Don't allow uploading of anything
- Don't allow uploading of scripts
- Provide some form of protection in browser

Disallowing Data Uploading

- Does your web site really need to allow users to upload stuff?

- Even if it does, must you show it to other users?

- If not, just don't take any user input

- **Problem**: Not possible for many important web sites

Don't Allow Script Uploading

- A no-brainer for most sites

  – Few web sites want users to upload scripts, after all

- So validate user input to detect and remove scripts

- **Problem**: Rich forms of data encoding make it hard to detect all scripts

- Good tools can make it easier

Protect the User's Web Browser

- Similar solutions as for any form of protecting from malicious scripts

- With the same problems:

  – Best solutions cripple functionality

- Firefox Content Security Policy

  – Allows web sites to specify where content can be loaded from

Cross-Site Request Forgery

- CSRF

- Works the other way around

- An authenticated and trusted user attacks a web server

  – Usually someone posing as that user

- Generally to fool server that the trusted user made a request

CSRF in Action

- Attacker puts link to (say) a bank on his web page

- Unsuspecting user clicks on the link

- His authentication cookie goes with the HTTP request

  – Since it's for the proper domain

- Bank authenticates him and transfers his funds to the attacker

Issues for CSRF Attacks

- Not always possible or easy

- Attacks sites that don't check referrer header

  - Indicating that request came from another web page

- Attacked site must allow web page to perform something useful (e.g., bank withdrawal)

- Must not require secrets from user

- Victim must click link on attacker's web site

- And attacker doesn't see responses

CSRF In the Wild

- CSRF in major Democratic Party fund raising site

- Cisco DNA Center (2021)
  - And Cisco's Data Center Manager in 2020

- WordPress had CSRF, too (2020)

- CSRF problem in Seagate Personal Cloud Drives (2018)

WordPress has replaced Adobe Flash as the poster child for poor security

Exploiting Statelessness

- HTTP is designed to be stateless

- But many useful web interactions are stateful

- Various tricks used to achieve statefulness

  - Usually requiring programmers to provide the state

  - Often trying to minimize work for the server

A Simple Example

- Web sites are set up as graphs of links
- You start at some predefined point
  - A top level page, e.g.
- And you traverse links to get to other pages
- But HTTP doesn't "keep track" of where you've been
  - Each request is simply the name of a link

Why Is That a Problem?

- What if there are unlinked pages on the server?

- Should a user be able to reach those merely by naming them?

- Is that what the site designers intended?

A Concrete Example

- The ApplyYourself system
- Used by colleges to handle student applications
- For example, by Harvard Business School in 2005
- Once all admissions decisions made, results available to students

What Went Wrong?

- Pages representing results were created as decisions were made

- Stored on the web server

    – But not linked to anything, since results not yet released

- Some appliers figured out how to craft URLs to access their pages

    – Finding out early if they were admitted

The Core Problem

- No protocol memory of what came before
- So no protocol way to determine that the request should be makeable
- Could be built into the application that handles requests
- But frequently isn't
  - Or is wrong

Solution Approaches

- Get better programmers

    – Or better programming tools

- Back end system that maintains and compares state

- Front end program that observes requests and responses

    – Producing state as a result

- Cookie-based

    – Store state in cookies (preferably encrypted)

Data Transport Issues

- The web is inherently a network application

- Thus, all issues of network security are relevant

- And all typical network security solutions are applicable

- Where do we see problems?

(Non-) Use of Data Encryption

- Much web traffic is not encrypted
  - Or signed
- As a result, it can be sniffed
- Allowing eavesdropping, MITM attacks, alteration of data in transit, etc.
- Why isn't it encrypted?

Why Web Sites Don't Use Encryption

- Primarily for cost reasons

- Crypto costs cycles

- For high-volume sites, not encrypting messages lets them buy fewer servers

- They are making a cost/benefit analysis decision

- And maybe it's right?

Problems With Not Using Encryption

- Sensitive data can pass in the clear
  - Passwords, credit card numbers, SSNs, etc.
- Attackers can get information from messages to allow injection attacks
- Attackers can readily profile traffic
  - Especially on non-secured wireless networks

Using Encryption on the Web

- Some web sites support use of HTTPS
  - Which permits encryption of data
  - Based on TLS/SSL

- Performs authentication and two-way encryption of traffic
  - Authentication is certificate-based

- HSTS (HTTP Strict Transport Security) <u>requires</u> browsers to use HTTPS

Increased Use of Web Encryption

- More major web sites are encrypting traffic
    - 97% of most popular sites default to HTTPS

- E.g., Google announced in 2014 it would encrypt all search requests
    - 95% of all Google traffic is HTTPS

- Facebook, Twitter adopted HSTS in 2014
    - Around 18% of all sites use it

- Arguably, <u>all</u> web interactions should be encrypted

Sometimes Encryption Isn't Enough

- Especially powerful "attackers" can subvert this process
  - Man-in-the-middle attacks by ISPs
  - NSA compromised key management
  - NSA also spied on private links
- Usually impossible for typical criminal
- Hard or impossible for a user to know if this is going on

Conclusion

- Web security problems not inherently different than general software security

- But generality, power, ubiquity of the web make them especially important

- Like many other security problems, constrained by legacy issues