CSC 405
Introduction to Computer Security

Web Security

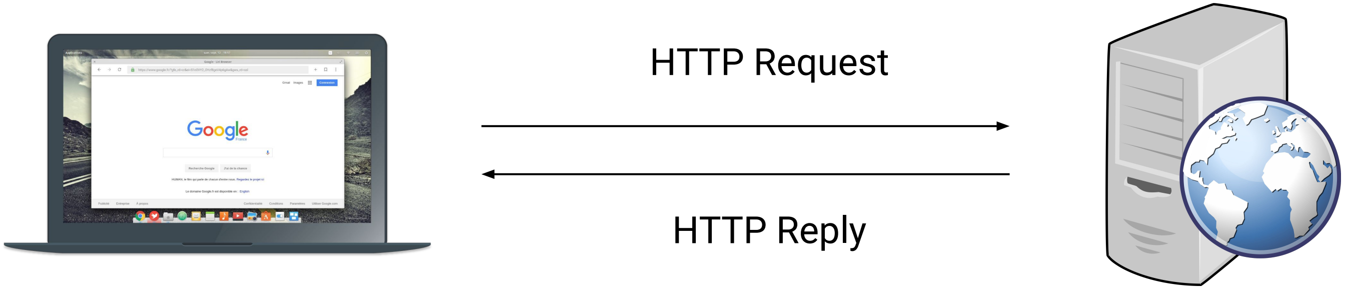
Alexandros Kapravelos
akaprav@ncsu.edu

(Derived from slides by Giovanni Vigna)

The World-Wide Web

- The World-Wide Web was originally conceived as a geographically **distributed document retrieval system** with a hypertext structure
- In the past 20+ years, the Web evolved into a full-fledged platform for the execution of distributed applications
- The Web is also vulnerable to a number of attacks
- The impact of these attacks is enormous, because of the widespread use of the service, the accessibility of the servers, and the widespread use of the clients

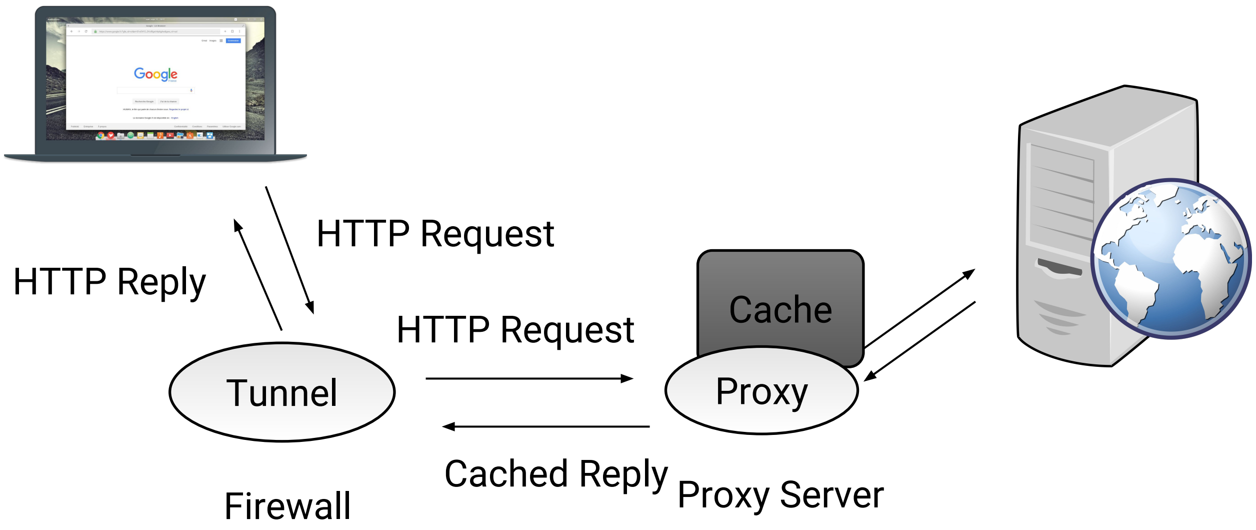
Architecture



HTTP Request

HTTP Reply

Architecture



HTTP Request

HTTP Reply

HTTP Request

Cache

Tunnel

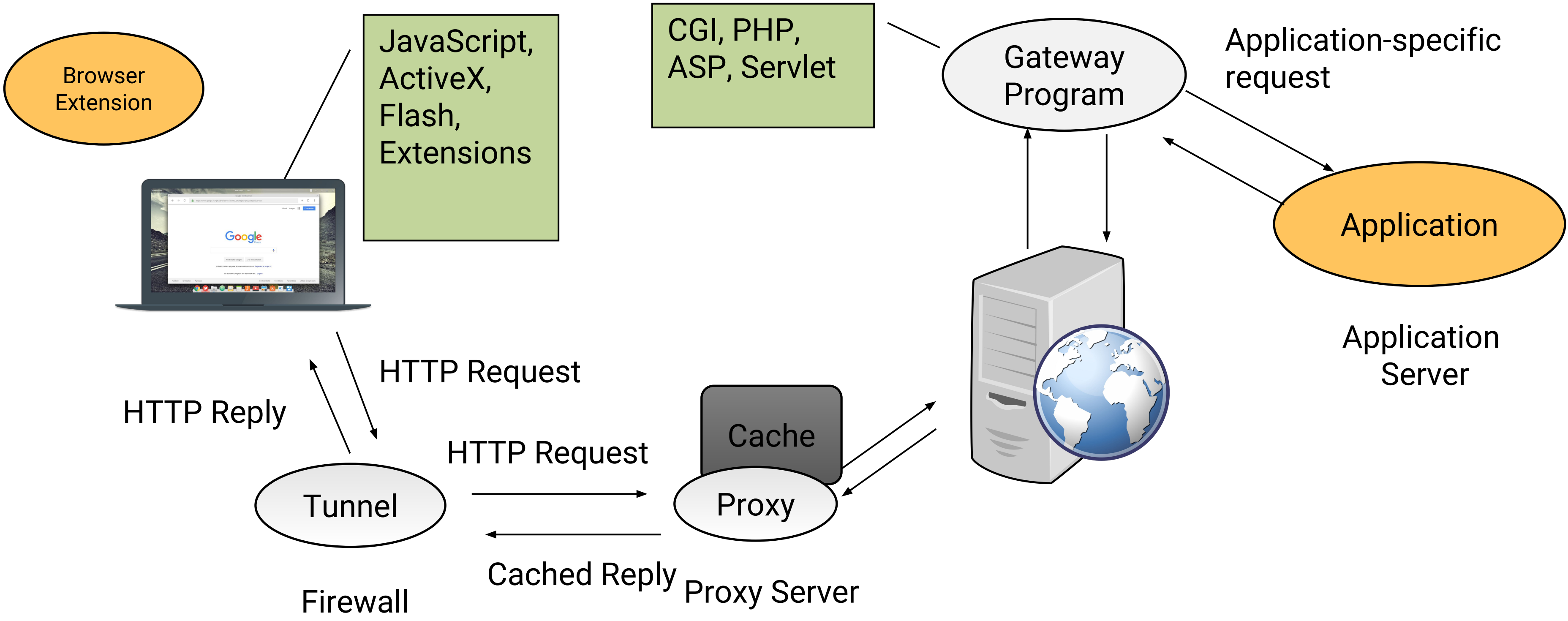
Proxy

Cached Reply

Proxy Server

Firewall

Architecture



Browser Extension

JavaScript, ActiveX, Flash, Extensions

CGI, PHP, ASP, Servlet

Gateway Program

Application-specific request

Application

Application Server

HTTP Request

HTTP Reply

HTTP Request

Cache

Tunnel

Proxy

Cached Reply

Firewall

Proxy Server

Standards and Technologies

- HTTP 1.0, 1.1
- URIs, URLs
- HTML, XML, XHTML
- DOM, BOM
- Cascading Style Sheets
- SSL/TLS, Socks
- CGI, Active Server Pages, Servlets
- JavaScript, VBScript
- Applets, ActiveX controls
- Web Services, SOAP

Web Vulnerability Analysis

- Vulnerabilities in the protocol(s)
- Vulnerabilities in the infrastructure
- Vulnerabilities in the server-side portion of the application
- Vulnerabilities in the client-side portion of the application
- Many vulnerability are the results of interactions of the various components involved in the processing of a request
- **Understanding the basic technologies is key**

Technology Review

- How are resources referenced?
- How are resources transferred?
- How are resources represented?
- How are resources processed on the server side?
- How are resources processed on the client side?

URIs, URLs, URNs

- Uniform Resource Identifier
  - a string that identifies a resource
- Uniform Resource Locator
  - an identifier that contains enough information to access the resource
- Uniform Resource Names
  - used to identify an entity regardless of the fact that the entity is accessible or even that it exists

URI Syntax

- The general URI syntax is specified in RFC 2396
- Specific types of URIs are described in separate standards
- Syntax: <scheme>://<authority><path>?<query>
- Examples:
  - ftp://ftp.ietf.org/rfc/rfc1808.txt
  - http://www.csc.ncsu.edu/~jdoe/My%20HomePage
  - mailto:cs176b@cs.csb.edu
  - telnet://melvyl.ucop.edu/

URI Syntax

- **Scheme**: a string specifying the protocol/framework
- **Authority**: a name space that qualifies the resource
  - Most of the times, it is a server name
    - <userinfo>@<host>:<port>
- **Path**: a pathname composed of “/” separated strings
- **Query**: an application-specific piece of information

HyperText Transfer Protocol

- Protocol used to transfer information between a web client and a web server
- Based on TCP, uses port 80
- Version 1.0 is defined in RFC 1945
- Version 1.1 is defined in RFC 2616

HTTP

- Client
  - Opens a TCP connection
  - Sends a request
- Server
  - Accepts the connection
  - Processes the request
  - Sends a reply
- Multiple requests can be sent using the same TCP connection

Requests

- A request is composed of a header and a body (optional) separated by an empty line (CR LF)
- The header specifies:
  - Method (GET, HEAD, POST)
  - Resource (e.g., /hypertext/doc.html)
  - Protocol version (HTTP/1.1)
  - Other info
    - General header
    - Request header
    - Entity header
- The body is considered as a byte stream

Methods

- **GET** requests the transfer of the entity referred by the URL
- **HEAD** requests the transfer of header meta-information only
- **POST** asks the server to process the included entity as “data” associated with the resource identified by the URL
  - Resource annotation
  - Message posting (newsgroups and mailing list)
  - Form data submission
  - Database input

Less-Used Methods

- **OPTIONS** requests information about the communication options available on the request/response chain identified by the URL (a URL of “*” identifies the options of the server)
- **PUT** requests that the enclosed entity be stored under the supplied URL (note that this is different from the POST request where the URL specifies the server-side component that will process the content)

Less-Used Methods

- **DELETE** requests that the origin server delete the resource identified by the URL
- **TRACE** invokes a remote, application-layer loop-back of the request message
  - TRACE allows the client to see what is being received at the other end of the request chain and use that data for testing or diagnostic information
- **CONNECT** is used with proxies

Resources

- A resource can be specified by an absolute URI or an absolute path
- Absolute URIs are used when requesting a resource through a proxy
  - `GET http://www.example.com/index.html HTTP/1.1`
- Absolute path URIs are used when requesting a resource to the server that owns that resource
  - `GET /index.html HTTP/1.1`

Request Example

```
GET /doc/activities.html HTTP/1.1
Host: longboard:8080
Date: Tue, 03 Nov 2015 8:34:12 GMT
Pragma: no-cache
Referer: http://www.ms.com/main.html
If-Modified-Since: Sat, 12 Oct 2016 10:55:15
GMT
<CR LF>
```

HTTP 1.1 Host Field

- In HTTP 1.0, it is not possible to discern, from the request line which server was intended to process the request:

  ```
  GET /index.html HTTP/1.0
  ```

- As a consequence it is not possible to associate multiple server "names" to the same IP address
- In HTTP 1.1, the "Host" field is REQUIRED and specifies which server is the intended recipient

  ```
  GET /index.html HTTP/1.1
  Host: foo.com
  ```

Replies

- Replies are composed of a header and a body separated by a empty line (CR LF)
- The header contains:
  - Protocol version (e.g., HTTP/1.0 or HTTP/1.1)
  - Status code
  - Diagnostic text
  - Other info
    - General header
    - Response header
    - Entity header
- The body is a byte stream

Status Codes

- 1xx: Informational - Request received, continuing process
- 2xx: Success - The action was successfully received, understood, and accepted
- 3xx: Redirection - Further action must be taken in order to complete the request
- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
- 5xx: Server Error - The server failed to fulfil an apparently valid request

Examples

- "200" ; OK
- "201" ; Created
- "202" ; Accepted
- "204" ; No Content
- “301" ; Moved Permanently
- "307" ; Temporary Redirect

- "400" ; Bad Request
- "401" ; Unauthorized
- "403" ; Forbidden
- "404" ; Not Found
- "500" ; Internal Server Error
- "501" ; Not Implemented
- "502" ; Bad Gateway
- "503" ; Service Unavailable

Reply Example

```
HTTP/1.1 200 OK
Date: Tue, 12 Oct 2016 8:35:12 GMT
Server: Apache/1.3.14 PHP/3.0.17 mod_perl/1.23
Content-Type: text/html
Last-Modified: Sun, 10 Oct 2016 18:11:00 GMT

<html>
  <head>
    <title>The Page</title>
  …
</html>
```

Header Fields

- General header fields: These refer to the message and not to the resource contained in it
  - Date, Pragma, Cache-Control, Transfer-Encoding..
- Request header fields:
  - Accept, Host, Authorization, From, If-modified-since, User Agent, Referer...
- Response header fields:
  - Location, Server, WWW-Authenticate
- Entity header fields:
  - Allow, Content-Encoding, Content-Length, Content-Type, Expires, Last-Modified

HTTP Authentication

- Based on a simple challenge-response scheme
- The challenge is returned by the server as part of a 401 (unauthorized) reply message and specifies the authentication schema to be used
- An authentication request refers to a realm, that is, a set of resources on the server
- The client must include an Authorization header field with the required (valid) credentials

HTTP Basic Authentication Scheme

- The server replies to an unauthorized request with a 401 message containing the header field

```
WWW-Authenticate: Basic realm=“ReservedDocs“
```

- The client retries the access including in the header a field containing a cookie composed of base64 encoded username and password

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

HTTP 1.1 Authentication

- Defines an additional authentication scheme based on cryptographic digests (RFC 2617)
  - Server sends a nonce as challenge
  - Client sends request with digest of the username, the password, the given nonce value, the HTTP method, and the requested URL
- To authenticate the users the web server has to have access to the hashes of usernames and passwords

Hypertext Markup Language

- A simple data format used to create hypertext documents that are portable from one platform to another
- Based on Standard Generalized Markup Language (SGML) (ISO 8879:1986)
- HTML 2.0
  - Proposed in RFC 1866 (November 1995)
- HTML 3.2
  - Proposed as World Wide Web Consortium (W3C) recommendation (January 1997)
- HTML 4.01
  - Proposed as W3C recommendation (December 1999)
- XHTML 1.0
  - Attempt by W3C to reformulate HTML into Extensible Markup Language (XML) (January 2000)
- HTML 5.0
  - Proposed as W3C recommendation (October 2014)
- HTML 5.1
  - Under development

HTML – Overview

- Basic idea is to “markup” document with tags, which add meaning to raw text
- Start tag: <foo>
- Followed by text
- End tag: </foo>
- Self-closing tag: <bar />
- Void tags (have no end tag): <img>
- Tag are hierarchical

HTML – Tags

```
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <p>I am the example text</p>
  </body>
</html>
```

HTML – Tags

- <html>
  - <head>
    - <title>
      - Example
  - <body>
    - <p>
      - I am the example text

HTML – Tags

- Tags can have “attributes” that provide metadata about the tag
- Attributes live inside the start tag after the tag name
- Four different syntax
  - <foo bar>
    - foo is the tag name and bar is an attribute
  - <foo bar=baz>
    - The attribute bar has the value baz
  - <foo bar='baz'>
  - <foo bar="baz">
- Multiple attributes are separated by spaces
  - <foo bar='baz' disabled required="true">

HTML – Hyperlink

- The anchor tag is used to create a hyperlink
- href attribute is used provide the URI
- Text inside the anchor tag is the text of the hyperlink

```
<a href="http://google.com">Google</a>
```

HTML – Basic HTML 5 Page

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>CS279</title>
  </head>

  <body>
    <a href="http://example.com/">Text</a>
  </body>
</html>
```

HTML – Character References

- Special characters can be included in HTML using < > ' " & =
  - Encode the character reference
  - Also referred to in HTML < 5.0 as “entity reference” or “entity encoding”
- Three types, each starts with & and ends with ;
  - Named character reference
    - &<predefined name>;
  - Decimal numeric character reference
    - &#<decimal unicode>;
  - Hexadecimal numeric character reference
    - &#x<hexadecimal unicode>;

HTML – Character References Example

- The ampersand (&) is used to start a character reference, so it must be encoded as a character reference
- &amp;
- &#38;
- &#x26;
- &#x00026;

HTML – Character References Example

- é
- &eacute;
- &#233;
- &#xe9;

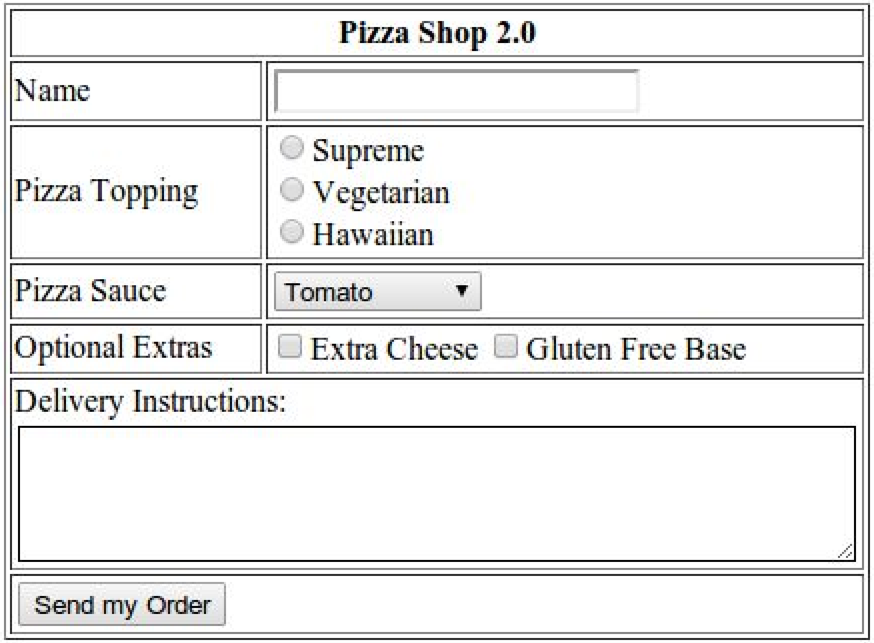
HTML – Character References Example

- &lt;
- &#60;
- &#x30;
- &#x00030;

HTML – Forms

- A form is a component of a Web page that has form controls, such as text fields, buttons, checkboxes, range controls, or color pickers
  - Form is a way to create a complex HTTP request
- The action attribute contains the URI to submit the HTTP request
  - Default is the current URI
- The method attribute is the HTTP method to use in the request
  - GET or POST, default is GET

HTML - Forms



Pizza Shop 2.0	
Name	
Pizza Topping	○ Supreme ○ Vegetarian ○ Hawaiian
Pizza Sauce	Tomato ▼
Optional Extras	□ Extra Cheese □ Gluten Free Base
Delivery Instructions:	
Send my Order	

HTML – Forms

- Children input tags of the form are transformed into either query URL parameters or HTTP request body
- Difference is based on the method attribute
  - GET passes data in the query
  - POST passes data in the body
- Data is encoded as either “application/x-www-form-urlencoded” or “multipart/form-data”
  - GET always uses “application/x-www-form-urlencoded”
  - POST depends on enctype attribute of form, default is “application/x-www-form-urlencoded”
  - "multipart/form-data" is mainly used to upload files

HTML – Forms

- Data sent as name-value pairs
  - Data from the input tags (as well as others)
    <input type="text" name="foo" value="bar”>
- Name is taken from the input tag’s name attribute
- Value is taken either from the input tag’s value attribute or the user-supplied input
  - Empty string if neither is present

application/x-www-form-urlencoded

- All name-value pairs of the form are encoded
- form-urlencoding encodes the name-value pairs using percent encoding
  - Except that spaces are translated to + instead of %20
  - foo=bar
- Multiple name-value pairs separated by ampersand (&)

application/x-www-form-urlencoded

```
<form action="http://example.com/grades/submit" >
  <input type="text" name="student" value="bar">
  <input type="text" name="class">
  <input type="text" name="grade">
  <input type="submit" name="submit">
</form>
```

```
http://example.com/grades/submit?student=John+Doe&class=csc+405&grade=A%2B&submit=Submit
```

application/x-www-form-urlencoded

```
<form action="http://example.com/grades/submit" method="POST">
  <input type="text" name="student”>
  <input type="text" name="class">
  <input type="text" name="grade">
  <input type="submit" name="submit">
</form>
```

```
POST /grades/submit HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:34.0)
Gecko/20100101 Firefox/34.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 65

student=John+Doe&class=CSC+405&grade=A%2B&submit=Submit
```

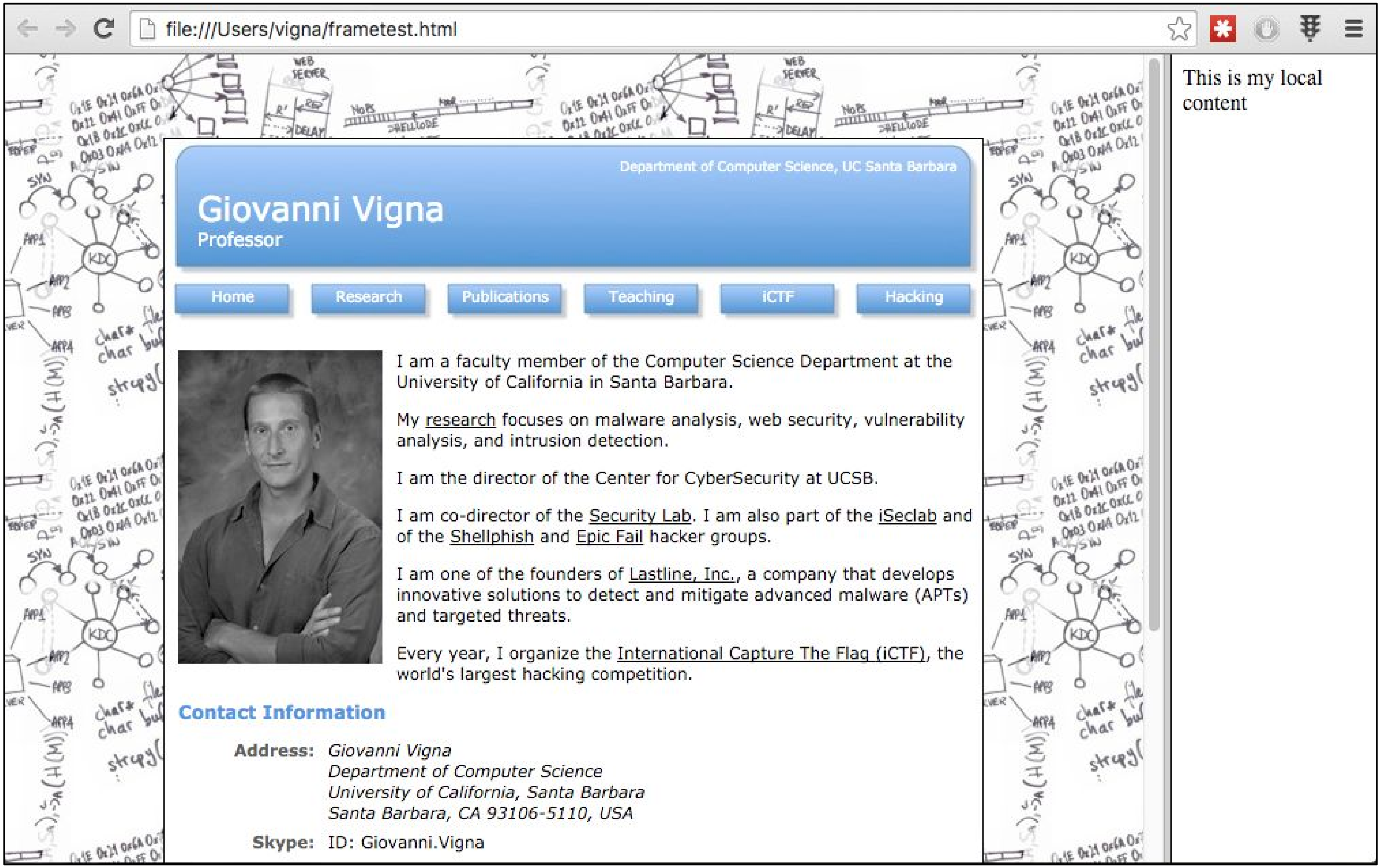
HTML Frames

- Frames allow for the display of multiple separate views (associated with separate URLs) together on one page
- Used in the early days to display banners or navigation elements
  - Now replaced by CSS directives

The frameset Element

```
<frameset cols="85%, 15%">
  <frame src="http://www.cs.ucsb.edu/~vigna" name="home">
  <frame src="frame.html" name="local">
  <noframes>
    Text to be displayed in browsers that do not support
frames
  </noframes>
</frameset>
```

The frameset Element



file:///Users/vigna/frametest.html

Department of Computer Science, UC Santa Barbara

Giovanni Vigna

Professor

I am a faculty member of the Computer Science Department at the University of California in Santa Barbara.

My research focuses on malware analysis, web security, vulnerability analysis, and intrusion detection.

I am the director of the Center for CyberSecurity at UCSB.

I am co-director of the Security Lab. I am also part of the iSeclab and of the Shellphish and Epic Fail hacker groups.

I am one of the founders of Lastline, Inc., a company that develops innovative solutions to detect and mitigate advanced malware (APTs) and targeted threats.

Every year, I organize the International Capture The Flag (iCTF), the world's largest hacking competition.

Contact Information

Address: *Giovanni Vigna*
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106-5110, USA

Skype: ID: Giovanni.Vigna

This is my local content

The iframe Element

- Inline frames
- Similar to frames, but does not need a frameset

```
<iframe src="http://www.kapravelos.com" name="home"
frameBorder="0"></iframe>

<iframe src="frame.html" name="frame"
frameBorder="0"></iframe>
```

Maintaining State

- HTTP is a stateless protocol
- Many web applications require that state be maintained across requests
- This can be achieved through a number of different means
  - Embedding information in the returned page
    - Modified URLs
    - Hidden fields in forms
  - Using cookies

Embedding Information in URLs

- When a user requests a page, the application embeds user-specific information in every link contained in the page returned to the user
- Client request:

```
GET /login.php?user=foo&pwd=bar HTTP/1.1
```

- Server reply:

```
<html>
...
<a href=“catalog.php?user=foo”>Catalog</a>
...
</html>
```

Embedding Information in Forms

- If a user has to go through a number of forms, information can be carried through using hidden input tags
- Client request:

```
GET /login.php?user=foo&pwd=bar HTTP/1.1
```

- Server reply:

```
<html>
... <form>
<input type=“hidden” name=“user” value=“foo” />
<input type=“submit” value=“Press here to see the catalog” />
...
```

- When the user presses on the form’s button, the string “user=foo” is sent together with the rest of the form’s contents

Embedding Information in Cookies

- Cookies are small information containers that a web server can store on a web client
- They are set by the server by including the “Set-Cookie” header field in a reply:

```
Set-Cookie: USER=foo; SHIPPING=fedex; path=/
```

- Cookies are passed (as part of the “Cookie” header field) in every further transaction with the site that set the cookie

```
Cookie: USER=foo; SHIPPING=fedex;
```

Embedding Information in Cookies

- They are usually used to maintain “state” across separate HTTP transactions
  - User preferences
  - Status of multi-step processes (e.g., shopping cart applications)
  - Session token stored as a result of a username/password authentication
- Cookies are accessible (e.g., through JavaScript) only by the site that set them

Cookie Structure

- A cookie can have a number of fields:
  - <name>=<value>: generic data (only required field)
  - expires=<date>: expiration date
  - path=<path>: set of resources to which the cookie applies
  - domain=<domain name>: by default set to the hostname, but it could specify a more generic domain (e.g., foo.com)
  - secure: flag that forces the cookie to be sent over secure connections only
  - httponly: flag that specifies that a cookie should not be accessible to client-side scripts
- There are limitations to the number of cookies that a server can set

Sessions

- Sessions are used to represent a time-limited interaction of a user with a web server
- There is no concept of a "session" at the HTTP level, and therefore it has to be implemented at the web-application level
  - Using cookies
  - Using URL parameters
  - Using hidden form fields
- At the beginning of a session a unique ID is generated and returned to the user
- From that point on, the session ID is used to index the information stored on the server side

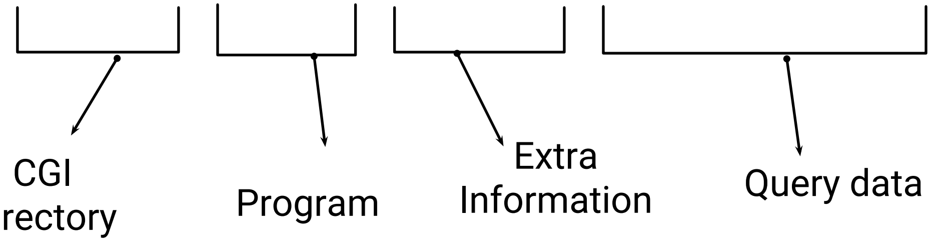
Executing Code on the Server

- The server-side component of an application executes code in reaction to an HTTP request
- This simple mechanism allows for the creation of web-based portal to database and other applications

The Common Gateway Interface

- Mechanism to invoke programs on the server side
- The program's output is returned to the client
- Input parameters can be passed
  - Using the URL (GET method)
    - Advantage: The query can be stored as a URL
  - Using the request body (POST method)
    - Advantage: Input parameters can be of any size

```
http://www.ms.com/cgi-bin/prg.exe/usr/info?choice=yes&q=high
```



CGI Directory

Program

Extra Information

Query data

CGI Programs

- Can be written in any language
- Input to the program is piped to the process' stdin
- Parameters are passed by setting environment variables
  - REQUEST_METHOD :GET, HEAD or POST
  - PATH_INFO : path in the URL that follows the program name and precedes "?"
  - QUERY_STRING: information that follows "?"
  - CONTENT_TYPE : MIME type of the data for the POST method
  - CONTENT_LENGTH : size of the data for the POST method
  - HTTP_<field>: value of corresponding header field

CGI Variables

- SERVER_SOFTWARE : name/version of server software
- SERVER_NAME : server hostname
- GATEWAY_INTERFACE : CGI version
- SERVER_PROTOCOL : server protocol version
- SERVER_PORT : TCP port used by the server
- PATH_TRANSLATED : PATH_INFO for non-Unix OSs
- SCRIPT_NAME : name of the script
- REMOTE_HOST : hostname of the client
- REMOTE_ADDR : address of the client
- AUTH_TYPE : authentication mechanism used
- REMOTE_USER : authenticated user name
- REMOTE_IDENT : user name as returned by identd

Active Server Pages

- Microsoft’s answer to CGI scripts
- Pages that contain a mix of
  - Text
  - HTML tags
  - Scripting directives (mostly VBScript and JScript)
  - Server-side includes
- Page scripting directives are executed on the server side before serving the page
- ASP.NET provide access to a number of easy-to-use built-in objects

Active Server Pages

```
<% strName = request.querystring("Name")
   If strName <> “” Then%>
<b>Welcome!</b>
<% Response.write(strName)
   Else %>
<b>You didn’t provide a name...</b>
<% End If %>
```

Servlets And JavaServer Pages (J2EE)

- Servlets are Java programs that are executed on the server
  - Similar to CGI programs
  - They can be executed within an existing JVM without having to create a new process
- JavaServer Pages (JSP) are static HTML intermixed with Java code
  - Similar to Microsoft’s Active Server Pages
  - Allow one to specify both the dynamic and the static parts of a page
  - They are compiled into servlets

PHP

- The “PHP Hypertext Processor” is a scripting language that can be embedded in HTML pages to generate dynamic content
- PHP code is executed on the server side when the page containing the code is requested
- A common setup is a LAMP system, which is the composition of
  - Linux
  - Apache
  - MySQL
  - PHP

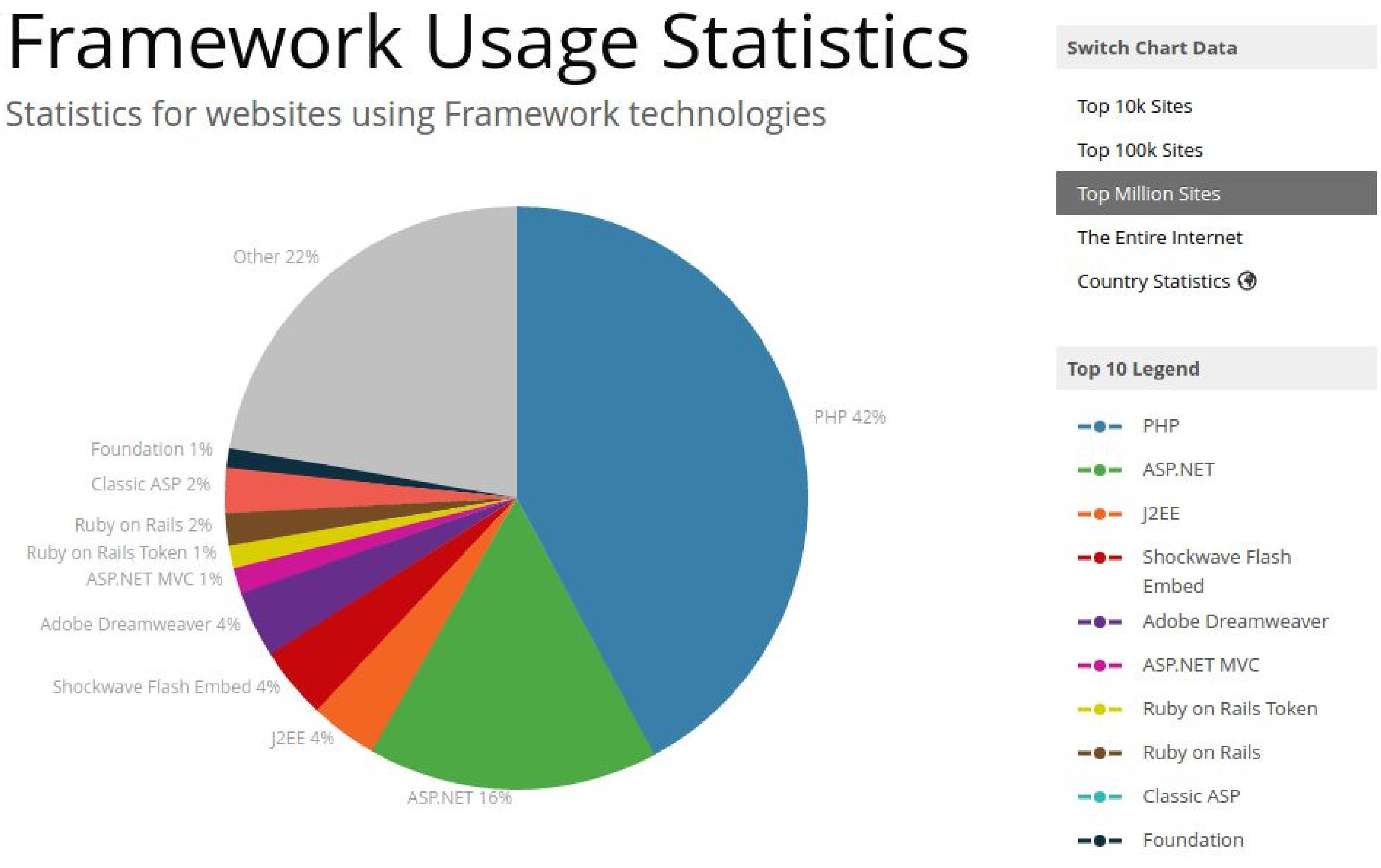
Example

```
<html>
  <head> <title>Feedback Page</title></head>
  <body>
    <h1>Feedback Page</h1>
    <?php
$name = $_POST['name'];
$comment = $_POST['comment'];
$file = fopen("feedback.html", "a");
fwrite($file, "<p>$name said: $comment</p>\n");
fclose($file);
include(”feedback.html");
    ?>
    <p>And this is the end of it!</p>
    <hr />
  </body>
</html>
```

Web Application Frameworks

- Web App Frameworks provide support for the rapid development of web applications
- Might be based on existing web servers or might provide a complete environment (including the server implementation)
- Often based on the Model-View-Controller architectural pattern
- Provide automated translation of objects to/from database
- Provide templates for the generation of dynamic pages
  - Ruby on Rails
  - Flask (Python)
  - Node.js (JavaScript)

Web Application Frameworks



Framework Usage Statistics

Statistics for websites using Framework technologies

Source: http://trends.builtwith.com/framework

User Agents

- User Agents (most of the time browsers) are the client side components responsible for the retrieval and display of web resources
  - wget, curl
  - Chrome, Firefox, Safari
- Some User Agents support the execution of client side code
  - Java Applets
  - ActiveX Controls
  - JavaScript

Java Applets

- Java applets are compiled Java programs that are
  - Downloaded into a browser
  - Executed within the context of a web page
- Access to resources is regulated by an implementation of the Java Security Manager
- Introduced in 1995, experienced initial success but was not adopted widely

ActiveX Controls

- ActiveX controls are binary, OS-specific programs that are downloaded and executed in the context of a web page
- ActiveX controls are supported only by Windows-based browsers
- The code is signed using the Authenticode mechanism
- Once executed, they have complete access to the client’s environment

JavaScript/JScript EcmaScript/VBScript

- Scripting languages used to implement dynamic behavior in web pages
- JavaScript initially introduced by NetScape in 1995 (LiveScript was the original name)
- JScript is Microsoft’s version (now also called JavaScript)
- EcmaScript is a standardized version of JavaScript
- VBScript is based on Microsoft Visual Basic

Client-side Scripting

- Code is included using external references

```
<script src=“http://www.foo.com/somecode.js”></script>
```

- Code is embedded into HTML pages using the SCRIPT tag and storing the code in comments

```
<script LANGUAGE=“JavaScript”>
<!-- var name = prompt ('Please Enter your name below.','')
   if ( name == null ) {
      document.write ('Welcome to my site!')
   }
     else {
      document.write ('Welcome to my site '+name+'!')
     }
-->
</script>
```

DOM and BOM

- The Document Object Model (DOM) is a programmatic interface to the manipulation of client-side content:

```
var x = document.createElement('HR');
document.getElementById('inserthrhere').appendChild(x);
```

- The Browser Object Model (BOM) is a programmatic interface to the browser properties:

```
location.href = 'newpage.html’;
history.back()
```

JavaScript Security

- JavaScript code is downloaded as part of an HTML page and executed on-the-fly
- The security of JavaScript code execution is guaranteed by a sandboxing mechanism
  - No access to files
  - No access to network resources
  - No window smaller than 100x100 pixels
  - No access to the browser’s history
  - ...
- The details of how sandboxing is implemented depend on the particular browser considered

JavaScript Security Policies (in Mozilla)

- “Same origin” policy
  - JavaScript code can access only resources (e.g., cookies) that are associated with the same origin (e.g., foo.com)
  - The protocol, port (if one is specified), and host are the same for both pages
- “Signed script” policy
  - The signature on JavaScript code is verified and a principal identity is extracted
  - The principal’s identity is compared to a policy file to determine the level of access
- “Configurable” policy
  - The user can manually modify the policy file (user.js) to allow or deny access to specific resources/methods for code downloaded from specific sites

Same Origin Policy In Detail

- Every frame in a browser’s window is associated with a domain
  - A domain is determined by the server, protocol, and port from which the frame content was downloaded
- Code downloaded in a frame can only access the resources associated with the source domain of the frame
- If a frame explicitly include external code, this code will execute within the frame domain even though it comes from another host

```
<script type="text/javascript"> //Downloaded from foo.com
      src="http://www.bar.com/scripts/script.js"> //Executes as if it were
   from foo.com
</script>
```

AJAX

- AJAX (Asynchronous JavaScript and XML) is a mechanism to modify a web page based on the result of a request, but without the need of user action
- It relies on two basic concepts:
  - JavaScript-based DOM manipulation
  - The XML-HTTP Request object

XML HTTP Request

- The XML HTTP Request object was introduced to allow JavaScript code to retrieve XML data from a server the execution of queries from JavaScript
- Unfortunately, the same object has to be accessed in different way depending on the browser being used
  - Most browsers:
    - `http_request = new XMLHttpRequest();`
  - Internet Explorer
    - `http_request = new ActiveXObject("Microsoft.XMLHTTP");`

Requesting A Document

- Using the “onreadystatechange” property of an XML-HTTP request object one can set the action to be performed when the result of a query is received

  -
    ```
    http_request.onreadystatechange = function(){
      code here
    };
    ```
- Then, one can execute the request

  -
    ```
    http_request.open('GET',
    'http://www.foo.com/show.php?keyword=foo', true);
    ```
  - Note that the third parameter indicates that the request is asynchronous, that is, the execution of JavaScript will proceed while the requested document is being downloaded

Waiting For The Document

- The function specified using the “onreadystatechange” property will be called at any change in the request status
  - 0 (uninitialized: Object is not initialized with data)
  - 1 (loading: Object is loading its data)
  - 2 (loaded: Object has finished loading its data)
  - 3 (interactive: User can interact with the object even though it is not fully loaded)
  - 4 (complete: Object is completely initialized)
- The function will usually wait until the status is “complete”
  -
    ```
    if (http_request.readyState == 4) {
      operates on data
    } else {
      not ready, return
    }
    ```

Modifying A Document

- After having received the document (and having checked for a successful return code -- 200) the content of the request can be accessed:
  - As a string by calling: http_request.responseText
  - As an XMLDocument object: http_request.responseXML
    - In this case the object can be modified using the JavaScript DOM interface

```
function reqListener () {
  console.log(this.responseText);
}

var oReq = new XMLHttpRequest();
oReq.addEventListener("load", reqListener);
oReq.open("GET", "http://example.com");
oReq.send();
```

Web Attacks

- Attacks against authentication
- Attacks against authorization
- Command injection attacks
- Unauthorized access to client information
- Man-in-the-middle attacks
- Attacks against HTTP protocol implementations

Monitoring and Modifying HTTP Traffic

- HTTP traffic can be analyzed in different ways
  - Sniffers can be used to collect traffic
  - Servers can be configured to create extensive logs
  - Browsers can be used to analyze the contents received from a server
  - Client-side/server-side proxies can be used to analyze the traffic without having to modify the target environment
- Client-side proxies are especially effective in performing vulnerability analysis of web applications because they allow one to examine and modify each request and reply
  - Burp
  - Chrome Postman Extension

Which Is The Best Way to Authenticate?

- IP address-based authentication
- HTTP-based authentication
- Certificate-based (SSL/TLS) authentication
- Form-based authentication

Web-based Authentication

- IP address-based
  - The IP source of a TCP connection (in theory) can be spoofed
  - NAT-ing may cause several users to share the same IP
  - The same user could use different IPs (for example, because of frequent DHCP renewals)
- HTTP-based
  - Not very scalable and difficult to manage at the application level
- Certificate-based
  - Works (on the server-side) for TLS-based connections
  - Few users have “real” certificates or know how to use them
- Form-based
  - Form data might be sent in the clear

Basic Authentication

- A form is used to send username and password (over an TLS-protected channel) to a server-side application
- The application:
  - Verifies the credentials (e.g., by checking in a database)
  - Generates a session authenticator which is sent back to the user
    - Typically a cookie, which is sent as part of the header, e.g.:
      `Set-Cookie: JSESSION=“johndoe:bluedog”; secure`
- Next time the browser contacts the same server it will include the authenticator
  - In the case of cookies, the request will contain, for example:
    `Cookie: auth=“johndoe:bluedog”`
- Authentication is performed using this value

Better Authentication

- Notes on previous scheme:
  - Authenticators should not have predictable values
  - Authenticators should not be reusable across sessions
- A better form of authentication is to generate a random value and store it with other session information in a file or back-end database
  - This can be automatically done using “sessions” in various frameworks
    - J2EE: JSESSIONID=1A530637289A03B07199A44E8D531429
    - PHP: PHPSESSID=43b4a19d1962304012a7531fb2bc50dd
    - ASP.NET: ASPSESSIONID=MBHHDGCBGGBJBMAEGLDAJLGF

Authentication Caveats

- If an application includes an authenticator in the URL it is possible that browsers may leak the information as part of the "Referer" [sic!] field
  - User access page http://www.foo.com/links.php?auth=28919830983
  - User selects a link to http://www.bar.com/
  - The www.bar.com site receives:

```
GET / HTTP/1.1
Host: www.bar.com
User-Agent: Mozilla
Referer: http://www.foo.com/links.php?auth=28919830983
```

More Caveats

- Authenticators should not be long-lived
- Note that a cookie's expiration date is enforced by the browser and not by the server
  - An attacker can manually modify the files where cookies are stored to prolong a cookie's lifetime
- Expiration information should be stored on the server's side or included in the cookie in a cryptographically secure way
- For example:
  - exp=t&data=s&digest=MACk(exp=t&data=s)

    see Fu et al. "Dos and Don'ts of Client Authentication on the Web"

Web Single Sign-On

- Authentication management can be a difficult task
- It is possible to rely on trusted third parties for authentication
  - OAuth
  - OpenId
  - SAML
  - FIDO

Attacking Authentication

- Eavesdropping credentials/authenticators
- Brute-forcing/guessing credentials/authenticators
- Bypassing authentication
  - SQL Injection
  - Session fixation

Eavesdropping Credentials and Authenticators

- If the HTTP connection is not protected by TLS it is possible to eavesdrop the credentials:
  - Username and password sent as part of an HTTP basic authentication exchange

```
05/12/05 11:03:11 tcp 253.2.19.172.in-addr.arpa.61312 ->
this.cs.ucdavis.edu 80 (http)
   GET /webreview/ HTTP/1.1
   Host: raid2005.cs.ucdavis.edu
   Authorization: Basic cmFpZGNoYWlyOnRvcDY4OQ== [raidchair:top688]
```

  - Username and password submitted through a form
  - The authenticator included as cookie, URL parameter, or hidden field in a form
- Cookies' “secure” flag is a good way to prevent accidental leaking of sensitive authentication information

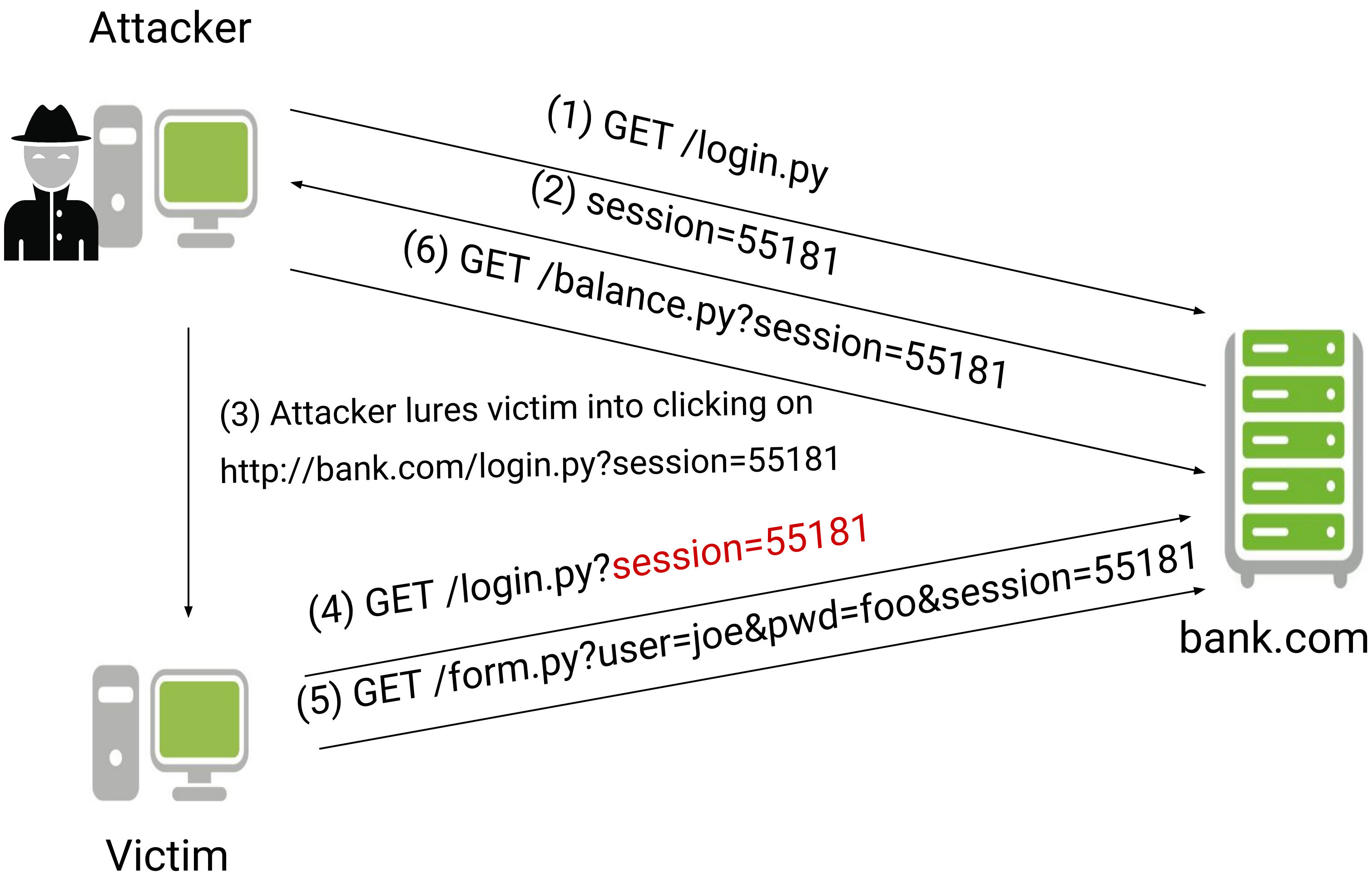
Brute-forcing Credentials and Authenticators

- If authenticators have a limited value domain they can be brute-forced (e.g., 4-digit PIN)
- If authenticators are chosen in a non-random way they can be easily guessed
  - Sequential session IDs
  - User-specified passwords
  - Example: http://www.foo.bar/secret.php?id=BGH15110915103939 observed at 15:11 of November 9, 2015
- Long-lived authenticators make these attacks more likely to succeed

Bypassing Authentication

- Form-based authentication may be bypassed using carefully crafted arguments (e.g., using SQL injection)
- Weak password recovery procedures can be leveraged to reset a victim’s password to a known value
- Authentication can be bypassed using forceful browsing
  - See discussion on authorization, later
- Authentication can be bypassed because of EAR
  - See discussion on EAR, later
- Authentication can be bypassed through session fixation

Session Fixation


Attacker
(1) GET /login.py
(2) session=55181
(6) GET /balance.py?session=55181
(3) Attacker lures victim into clicking on
http://bank.com/login.py?session=55181
(4) GET /login.py?session=55181
(5) GET /form.py?user=joe&pwd=foo&session=55181
bank.com
Victim

Session Fixation

- If application accepts blindly an existing session ID, then the initial setup phase is not necessary
- Session IDs should always regenerated after login and never allow to be “inherited”
- Session fixation can be composed with cross-site scripting to achieve session id initialization (e.g., by setting the cookie value)

- See: M. Kolsek, “Session Fixation Vulnerability in Web-based Applications”

Lessons Learned

- Authentication is critical
- Do not transfer security-critical information in the clear
- Do not use repeatable, predictable, long-lived session IDs
- Do not allow the user to choose the session IDs
- If possible, use well-established third-party authentication services

Authorization Attacks: Forceful Browsing

- Resources in a web application are identified by paths
- The web application developer assumes that the application will be accessed through links, following the “intended flow”
- The user, however, is not bound to follow the prescribed links and can “jump” to any publicly available resource
- If paths are predictable, one can bypass authorization checks
- Example:
  - User is presented with list of documents only after authentication
  - Requesting directly the URL http://www.acme.com/resources/ provides access

Authorization Attacks: Path Traversal

- Applications might build filename paths using user-provided input
- Path/directory traversal attacks
  - Break out of the document space by using relative paths
    - `GET /show.php?file=/../../../../../../etc/passwd`
    - Paths can be encoded, double-encoded, obfuscated, etc
    - `GET show.php?file=%2f%2e%2e%2f%2e%2e%2fetc%2fpasswd`

Authorization Attacks: Directory Listing

- If automated directory listing is enabled, the browser may return a listing of the directory if no index.html file is present and may expose contents that should not be accessible

Lesson Learned

- Resources are identified by paths
  - Web pages
  - Filenames
- If the resources identifiers are predictable, it is possible to bypass authorization checks