# CSC 574
# Computer and Network Security

# Web Security

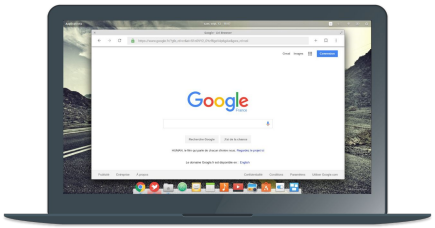
Alexandros Kapravelos
kapravelos@ncsu.edu

(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

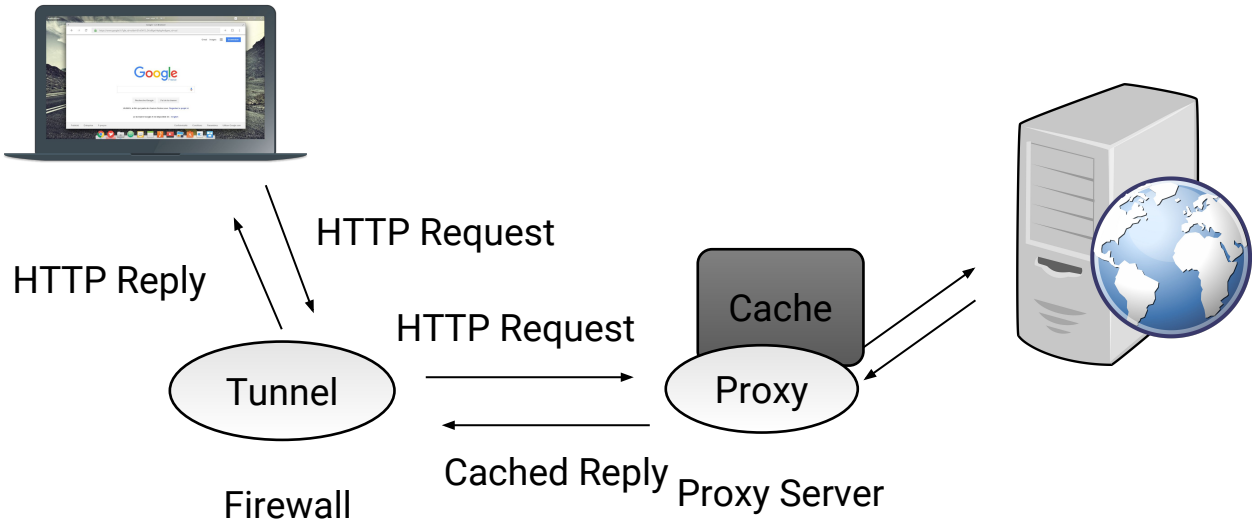
# Architecture



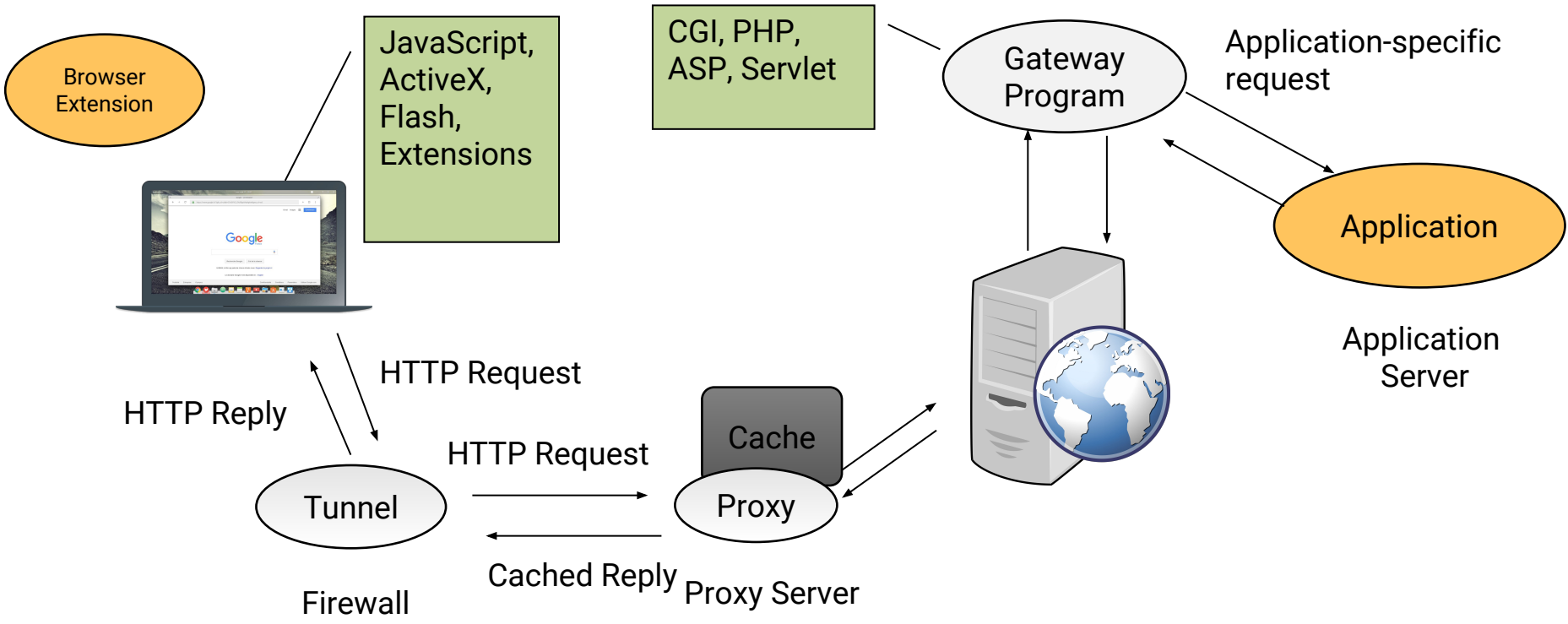

HTTP Request

HTTP Reply

# Architecture



# Architecture



# 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
- &
- &
- &
- &

# HTML – Character References Example

- é
- é
- é
- é

# HTML – Character References Example

- <
- <
- 0
- 0

# 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

- 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=cs
+279&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=CS+279&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>
```