

University of Rochester  
CSC290B  
Introduction to Computer Security

Web Security Mechanisms

March 5, 2009

# Browser Basic Concepts

- Users use browsers that make requests to servers.
- Browsers support many protocols
  - But Hypertext Transfer Protocol (HTTP) is predominant.
- Browsers support many data types
  - But Hypertext Metalanguage (HTML) is predominant
- A single request may result in a flurry of subsequent protocol activity

# Hypertext Transfer Protocol (HTTP)

- Request/response protocol that negotiates transfer features and content type, and performs content transfer.

– Example request/response:

- `POST /fuzzy_bunnies/images/bunny_dispenser.php HTTP/1.1`  
`Host: www.fuzzybunnies.com`  
`User-Agent: Bunny-Browser/1.7`  
`Content-Type: text/plain`  
`Content-Length: 12`  
`Referer: http://www.fuzzybunnies.com/main.html`

`HELLO SERVER`

- `HTTP/1.1 200 OK`  
`Server: Bunny-Server/0.9.2`  
`Content-Type: text/plain`

`HELLO CLIENT`

# Browser State

- HTTP protocol is basically stateless.
- Applications built on top of HTTP must devise their own mechanisms for establishing state, e.g. to create server-side security context or browser-side security context

# HTTP Authentication

- Allows HTTP Server to challenge client for a credential (e.g. username and password)
- Client caches the credential and replays it for HTTP server
- Credentials:
  - Requested/required by server
  - Generated by *client*
  - Provided by client

# Cookies

- Cookies implement one mechanism:
  - Small persistent storage  
name/value/expiration storage, data set  
by servers in HTTP responses
  - Data returned by clients in subsequent  
HTTP requests
- Credentials:
  - Requested/required by server
  - Generated by *server*
  - Provided by client

# Attacks on Browser State

- Predict/guess/search for state, forge credentials
- Steal credentials
- Convince browser to use the credentials it already has

# Document Object Model (DOM)

- HTML documents are programatically accessible and modifiable on the fly

# Browser-side Javascript

- Javascript a.k.a. ECMAScript
  - Object-based scripting language tightly integrated with HTML and DOM
  - Can be invoked in a variety of ways
    1. Standalone `<SCRIPT>` tags that enclose code blocks,
    2. Event handlers tied to HTML tags (e.g. `onmouseover="..."`),
    3. Stylesheet expression(...) blocks that permit Javascript syntax in some browsers,
    4. Special URL schemes specified as targets for certain resources or actions (`javascript:...`).

# Browser-side Security Context

- Scripting language implementation has access to all internal state of the browser.
- Can this dynamic code cause the browser to do anything that a document's script tells it to do?
  - Every action invoked by a browser has a security context used to enforce rules about what the browser will allow

# Same-origin policy

- Many browser security rules are based on the same-origin policy: that access (read or modify) is restricted to only the originator of an object
  - DOM: Scripts can only access documents from the same origin.
  - Cookies: Scripts can only access cookies from the same origin.
  - Network connections can only be made to the server of origin (Java applets, Flash, XMLHttpRequest)

# It depends on what the meaning of “same” is

- Cooperating subdomains
  - example.com
  - www.example.com
  - payments.example.com
- Note, if name lookups are compromised, same-origin policy is compromised
  - Local host tables
  - DNS rebinding attacks

# Cross-site scripting (XSS)

- Same-origin policy assumes that the server is controlling or vetting the dynamic content
- If user- or third-party-provided content is replayed by a server, then you have executable scripts provided by a (potentially hostile) user executing with a security context established by the server

# Cross-site Request Forgery (CSRF)

- Browsers always allow users to follow links, with just a click (at most), with context where same-origin policy does not apply.
- If a web application's actions use known URLs, plus some client state the browser will always supply the server (e.g. cookies), then
  - Hostile web page (from anywhere) can cause the user to invoke a web application's actions