



Stony Brook University

CSE 361: Web Security

Basic Client-Side Technologies/Security

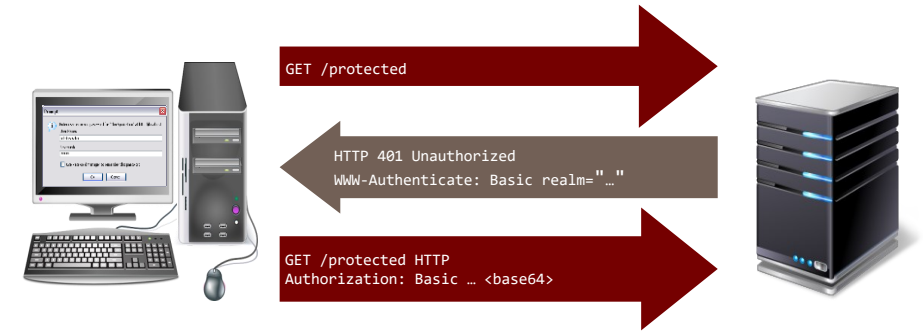
Nick Nikiforakis

Adding State to HTTP

- Recall: no inherent state in HTTP
 - server does not keep any state after TCP connection is closed
- For static content sites, no problem
 - developing "applications" is impossible though
 - e.g., shopping cart on Amazon
- Need to introduce state in HTTP
 - in the form of "sessions"

Option 1: HTTP Authentication

- Associate user with state on server
 - unclear when the "sessions" ends
- Authentication done by Web server
 - Not by the application served via the server
- Implements "pulling" of credentials
 - User: "Please give me resource X"
 - Server: "No, please tell me who you are"
 - User: "Ok, I am *alice* and my password is *nu7^yjUtasw* "
- Logout non-trivial
 - browser always sends along authentication header



Option 2: Session Identifier in URL

- Generate random token on first page visit
- Ensure that session ID is in all links
- Potential for accidental leakage is high
 - "Here is the link to the product on Amazon"
- URL is transmitted in Referer header
 - Session leaked to all included third-party sites

```
http://example.org/  
cart.html?sess=9b2dac168331
```



Option 3: Cookies

- Generate random token on first page visit
- Sent to client via Set-Cookie header
- Client always sends along cookies in every request to the server
 - important: regardless of initiating site
- Cookies are persisted in the browser
 - controllable by Expires option in cookie
 - default: delete on session end (when browser is closed)
- Ending session: delete cookie



Cookie directives

- `<name>=<value>`
- `Expires=<Date>`, determines when cookie should be deleted
- `Max-Age=<Seconds>`, determines when cookie should be deleted
- `Domain=<domain>`, defaults to current host
 - Can be set for parent domains (and their subdomains)
 - If nothing is specifically set, cookie is only set for current domain without subdomains
 - `Domain=example.com` on `websec.example.com` sets cookie for `*.example.com` and `example.com`
- `Path=<path>`, only set cookie for this path (and sub-paths)

Cookie directives

- `HttpOnly`, disallows access from JavaScript via `document.cookie`
- `Secure`, only transmit cookie over secure connection
 - Can only be set from HTTPS connections
- `SameSite=None/Strict/Lax`
 - `Strict`: do not transmit cookies on **any** cross-site request
 - `Lax`: only transmit cookies on "safe" top-level navigation
 - Safe methods (per RFC 7231): GET, HEAD, OPTIONS, (TRACE)
 - `None`: explicit opt-in for cross-site requests, requires `Secure`
 - Browsers will default to `SameSite=Lax` soon (Chrome already does so, FF and Edge warn)

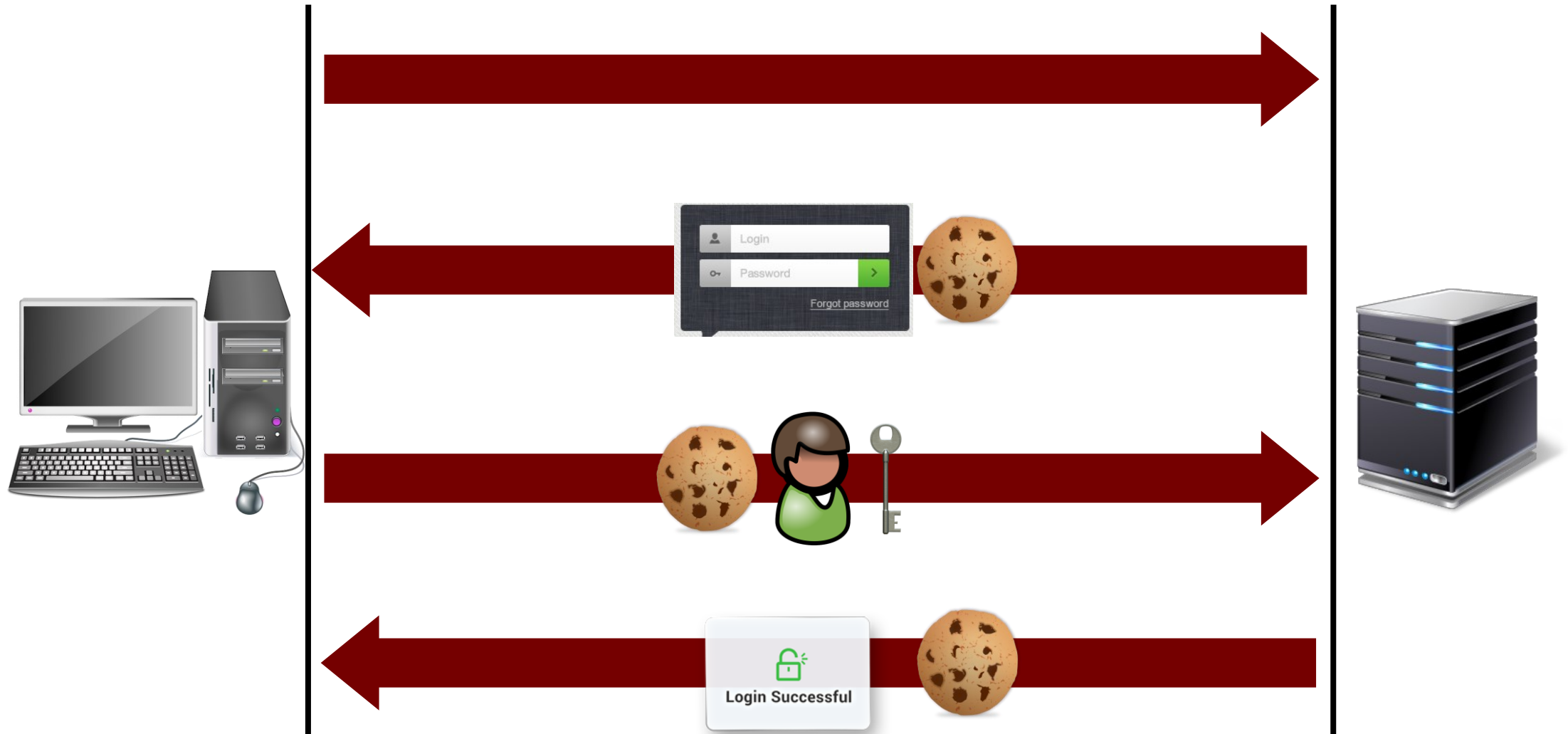
Cookie examples

- `Set-Cookie: test=1; Domain=.example.com; Secure; HttpOnly; SameSite=None`
 - Sets a cookie with name "test" to the value "1"
 - Cookie will be sent to any HTTPS request made to example.com and any subdomain
 - Cookie is not accessible from JavaScript
 - Cookie will be sent on cross-site requests as well
 - Cookie will be deleted on browser close (no explicit expiry date)
- `Set-Cookie: test=1; Domain=.example.com; HttpOnly; SameSite=None`
 - Chrome will not accept this (SameSite=None requires Secure)
 - FF and Edge will warn

Form-based authentication

- Default today: HTML forms
 - Server provides form with username and password fields
 - User fills and submits form
 - Server decides if credentials were correct, and "upgrades" session
 - actually better: create new session (more on that later)
- Password fields hide input with ***
 - besides this, not different than any other input field
 - accessible via JavaScript
 - sent in clear text via GET or POST to server
 - can be sent cross-domain (a.com can send data to b.com)

Form-based authentication



Authentication with cookies - caveats

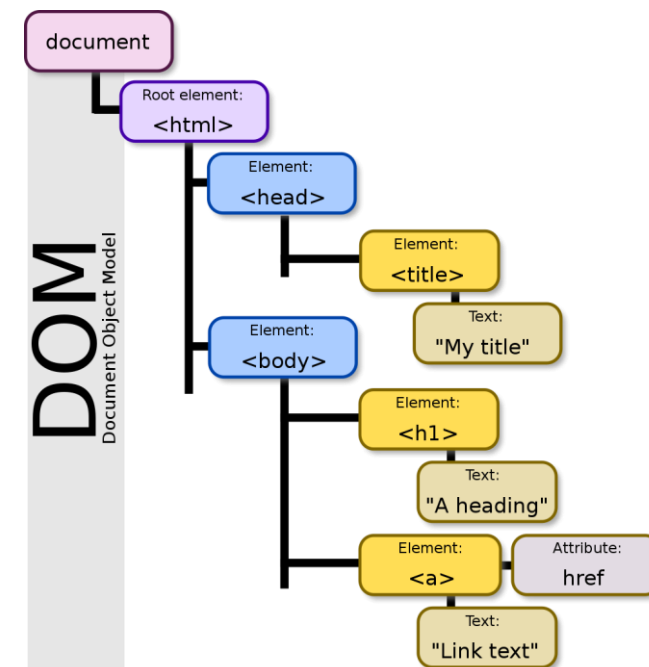
- Cookies were not designed with security in mind
 - cookies readable and writeable from JavaScript (unless HttpOnly is used)
 - if set for a given domain, valid for all sub-domains
 - added to all requests, regardless of the origin of requesting site
- Several security problems from this (which we cover later)
 - Session Hijacking
 - Session Fixation
 - Cross-Site Request Forgery
 - Cross-Site Script Inclusion

JavaScript

```
var simple = type  
forward = type.slice(  
ofType = what === "of-type";  
  
return first === 1 && last === 0 ?  
  
// Shortcut for :nth-*(n)  
function( elem ) {  
    return !!elem.parentNode;  
} :  
  
function( elem, context, xml ) {  
    var cache, outerCache, node, diff, nodeIndex, start,  
        dir = simple !== forward ? "nextSibling" : "  
        previousSibling",  
        parent = elem.parentNode,  
        name = ofType && elem.nodeName.toLowerCase(),  
        useCache = !xml && !ofType;  
  
    if ( parent ) {
```

What is JavaScript in the browser?

- JavaScript core
 - ECMAScript specified language
 - initially developed for Netscape in 1995 as LiveScript/JavaScript
- The Document Object Model (DOM)
 - provides access to the rendered HTML document
 - allows controlling the browsing window via JavaScript
- Browser-based standard APIs
 - Math, WebStorage, XMLHttpRequest, ...



JavaScript Core

- Functional programming language
 - object model is prototype-based
 - no class hierarchy
 - allows for closures and anonymous functions
- No native concurrency model
 - JavaScript in an execution context (e.g., a Web document) is single-threaded
 - Concurrency is event-driven
 - Do something, yield process, wait for wake-up
 - e.g., implemented by setTimeout with (potentially anonymous) callback function
 - loading the same page twice might not execute instructions in the same order

JavaScript in Web documents

- JavaScript can be included in script tags or event handlers
 - `<script>var hello="world";</script>`
 - `<script src="http://hello.world"></script>`
 - `Click me`
- Each script tag or event handler is separate parsing block
 - code not executed when parsing error occurs
 - other scripts' execution is not interrupted
- Rendering of document stops until script is executed
 - especially important when HTML is written by JavaScript
- **All scripts run in same global space (of including page)**

JavaScript Objects

- JavaScript is highly flexible
 - Dynamic typing at its best
 - Lots of implicit type casting
 - "a" + 1 => "a1"
 - "a" + undefined => "aundefined"
 - alert(42) => alert(42.toString())
- Primitives types (strings, numerical, ..) and Object types
- New properties can be added to existing objects

```
var myObj = new myObject();  
myObj.a = 1;
```


JavaScript Prototype-based Object Model

- All objects have a *prototype*
 - Prototype can have prototype as well
 - so-called prototype chaining
- Function call is propagated along chain until either
 - corresponding function is found
 - prototype is null (for Object)

```
var a = "a";  
a.__proto__  
// > String {length: 0, constructor: function,...}  
a.__proto__.__proto__  
// > Object {__defineGetter__: function, ...}  
a.__proto__.__proto__.__proto__  
// > null
```

JavaScript Prototype-based Object Model

- Prototypes can be set and manipulated during runtime
- Prototype changes also affect existing objects

```
Number.prototype.toString = function() {  
    return "Gotcha";  
};  
  
// This will display "Gotcha" instead of 42  
alert(new Number(42));
```

```
var fortytwo = new Number(42);  
// This will display "42"  
alert(fortytwo);  
Number.prototype.toString = function() {  
    return "Gotcha again";  
};  
  
// This will display "Gotcha again"  
alert(fortytwo);
```

JavaScript Objects

- Objects are instances of functions

```
function myObj(p1, p2) {  
  this.m1 = p1;  
  this.m2 = p2;  
}  
var x = new myObj(1,2);  
// > myObj {m1: 1, m2: 2}
```

- Also true for built-in objects

```
Number  
// > function Number() { [native code] }  
Number.constructor  
// > function Function() { [native code] }
```

- Almost everything has a toString()

```
myObj.toString()  
"function myObj(p1, p2) {  
  this.m1 = p1;  
  this.m2 = p2;  
}"
```

JavaScript Variable Scoping

- Variables without `var` keyword always in global scope
- Variables with `var` keyword as specified in current scope (function-level)
 - Gotcha: in top-level script code, that is the global scope
- Public members of object use `this` keyword, private members `var`

```
function Container(param) {  
    var member = param;  
}
```

```
var a = new Container(1);  
a.member  
// > undefined
```

```
function Container(param) {  
    this.member = param;  
}
```

```
var a = new Container(1);  
a.member  
// > 1
```

```
function Container(param) {  
    var member = param;  
    this.getmember = function() {  
        return member; }  
}
```

```
var a = new Container(1);  
a.getmember()  
// > 1
```

Getters, Setters, and Freezing

- ECMAScript introduced the `Object.defineProperty` method
 - `get` and `set` to allow read/write access to properties
 - `configurable` to prevent redefinition for the property

```
var obj = new Container(1);
var mValue = 42;

Object.defineProperty(obj, "member", {
  get: function() { return mValue; },
  set: function(newValue) { mValue = newValue; },
  configurable: false});

obj.member
// > 42
obj.member = 43
mValue
// > 43
Object.defineProperty(obj, "member", {get: function() { return 1; }});
// > Uncaught TypeError: Cannot redefine property: member
```

(Almost) everything in JavaScript can be overwritten/deleted

```
eval("var a='hello'")
a
// > "hello"

eval = alert;

eval("var a='hello'");
// opens alert box
```

```
var oAlert = alert;
alert = function(x) {
  console.log(x);
  oAlert(x);
}
alert(1);
// log 1 to console
// opens alert box
```

```
var oAlert = alert;
delete alert;

alert(1);
// Uncaught ReferenceError: alert is not defined

oAlert(1)
// opens alert box
```

Document Object Model (DOM) and Browser APIs

- Exposed to JavaScript through global objects
 - `document`: Access to the document (e.g., cookies, head/body)
 - `navigator`: Information about the browser (e.g., UA, plugins)
 - `screen`: Information about the screen (e.g., dimension, color depth)
 - `location`: Access to the URL (read and modify)
 - `history`: Navigation
- Global object is called `window`, current object is `self`

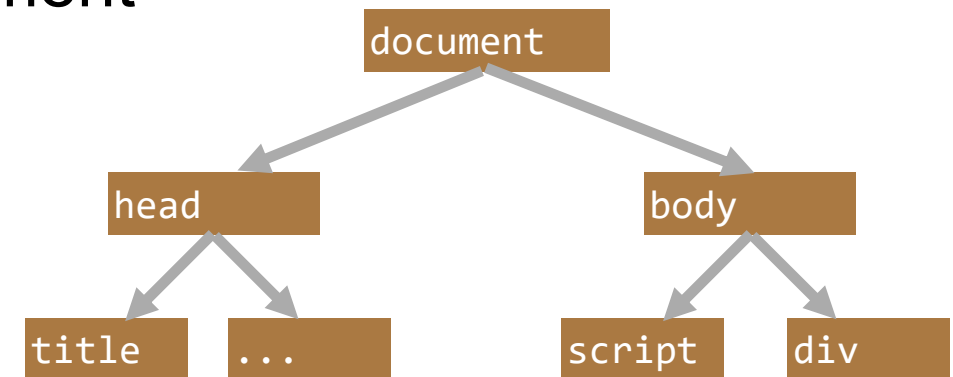
```
a = "Hello";  
a === window.a;  
> true
```

```
document.location === location;  
> true
```

```
self === window;  
> true
```

Manipulating the rendered document

- HTML represented by a tree of HTML`Element` objects
- Element attributes of HTML nodes map to properties of HTML`Element` object
 - `document.body.children[1].style.color = "red"`
- Several methods/properties to change document
 - `document.write`
 - `element.innerHTML/element.outerHTML`
 - `element.attribute`
 - `element.appendChild`
- Elements with `id` automatically in global scope

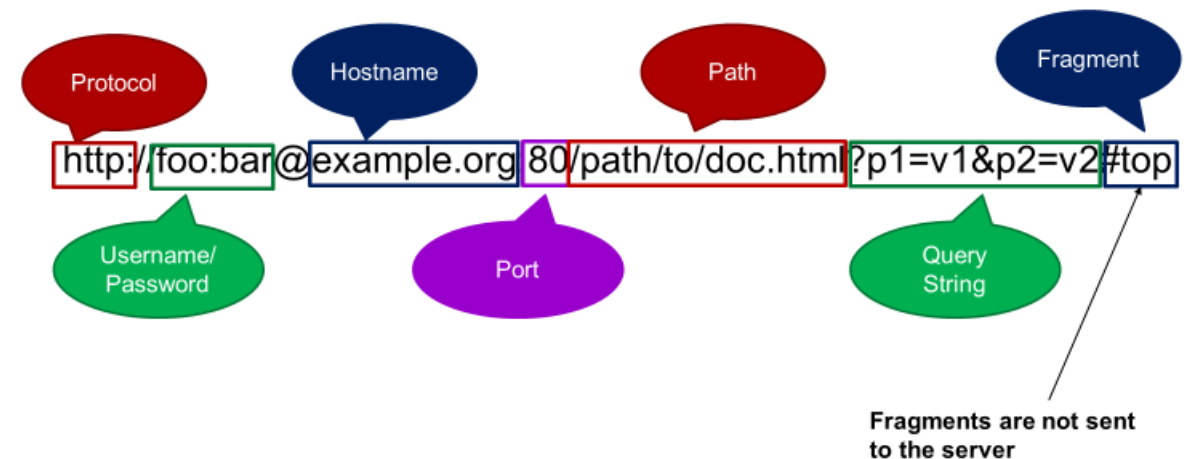


Access to other documents

- Handles to other frames in same browsing window
 - `parent`
 - `top`
 - `frames[]`
- Handles to popup windows
 - `var handle = window.open("http://example.org")`
 - `window.opener`
- Initially no security considerations...

The location object

- `location.href`: complete URL including fragment
- `location.host`: HTTP host, including port (if any)
`location.hostname`: only HTTP host
`location.port`: only the port (if non-standard)
- `location.protocol`: protocol
- `location.pathname`: path
- `location.search`: URL query
- `location.hash`: URL fragment




Summary

5

Option 3: Cookies

- Generate random token on first page visit
- Sent to client via Set-Cookie header
- Client always sends along cookies in every request to the server
 - important: regardless of initiating site
- Cookies are persisted in the browser
 - controllable by Expires option in cookie
 - default: delete on session end (when browser is closed)
- Ending session: delete cookie




15

JavaScript in Web documents

- JavaScript can be included in script tags or event handlers
 - `<script>var hello="world";</script>`
 - `<script src="http://hello.world"></script>`
 - `Click me`
- Each script tag or event handler is separate parsing block
 - code not executed when parsing error occurs
 - other scripts' execution is not interrupted
- Rendering of document stops until script is execute
 - especially important when HTML is written by JavaScript
- **All scripts run in same global space (of including page)**

10

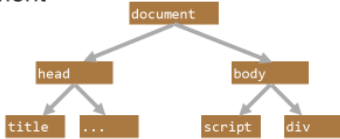
Form-based authentication



24

Manipulating the rendered document

- HTML represented by a tree of HTML Element objects
- Element attributes of HTML nodes map to properties of HTML Element object
 - `document.body.children[1].style.color = "red"`
- Several methods/properties to change document
 - `document.write`
 - `element.innerHTML/element.outerHTML`
 - `element.attribute`
 - `element.appendChild`
- Elements with id automatically in global scope



Credits

- Original slide deck by Ben Stock
- Modified by Nick Nikiforakis