Imperial College London

Network and Web Security

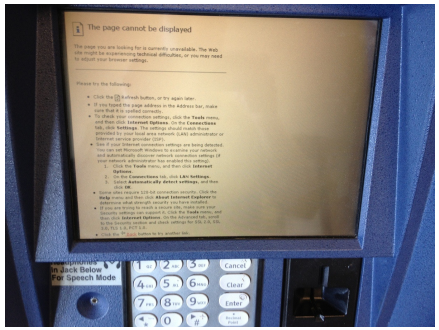
Browser security

Dr Sergio Maffeis
Department of Computing
Course web page: https://331.websec.fun

The browser

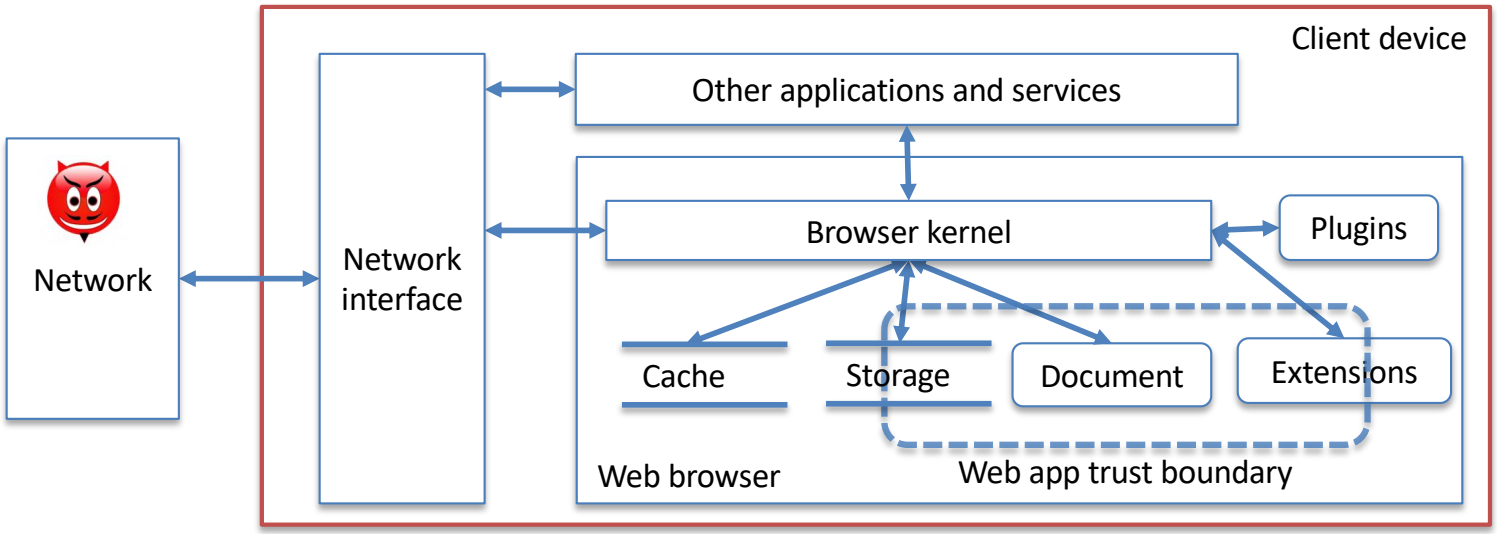
Imperial College London

- Browsers are everywhere: PCs, phones, cars, planes, ATMs, …
- Main goal: enable the user browse the web
  - Functionality and compatibility above all
  - For individual vendors: market share too
- Main security concerns
  - Protect the device hosting the browser from infection
  - Protect user data from unauthorized access
  - Provide a trustworthy platform to deploy the client-side of a web application
    - Web apps should be as secure as desktop apps
- To fully appreciate the rest of the course, you should familiarize with
  - Standard browser technologies
    - HTTP, HTML, CSS, JavaScript, DOM, AJAX
  - Google Chrome
    - Most used browser: overall, and also by people attending this class
    - We use it in our examples, but concepts apply to other browsers as well
      - With notable exceptions for various IE versions
  - Chrome Developer Tools: https://developer.chrome.com/devtools

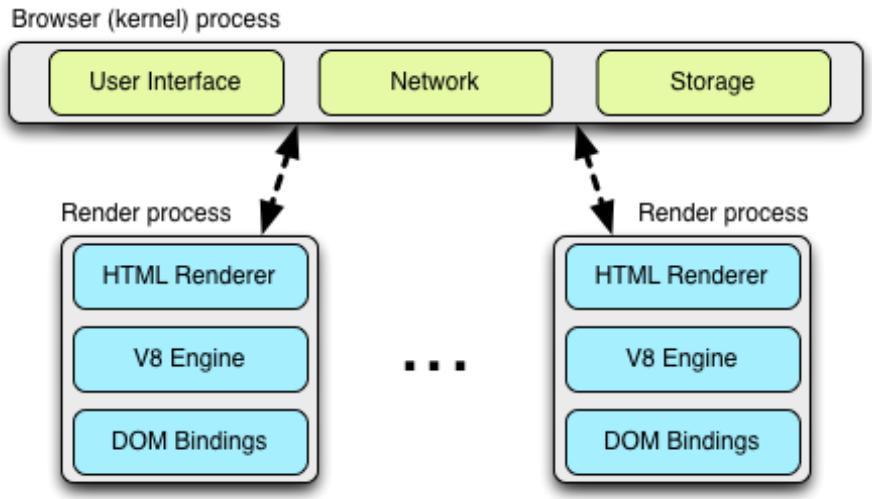


Client architecture

Imperial College London



- Chrome browser
  - Leverages OS process-based isolation and sandboxing to limit effects of compromise
  - Each window/tab has its own process with renderer, JS engine, DOM
  - Efficient networking architecture
    - Socket reuse
    - Predictive optimizations



(ivgita.com)

Chrome's chrome

Imperial College London

- The *chrome* is also the user interface of a browser
  - Location bar, browser dialogs, settings, bookmarks
  - Cannot be tampered with by web pages
  - Aims to be spoof-resistant

Attacks: phishing

Imperial College London

- Goal of the attacker is to steal user credentials or other sensitive information
- User visits a page controlled by the attacker that looks like a page form the target
  - Full replica of target HTML
  - Screenshot of target page, plus scripts that simulate interactive behaviour (like forms)
  - Different page that imitates target branding only
- User reveals data to the attack page
- Attack page forwards data to attacker and to legitimate page, displays error message, or short acknowledgement
- Hosting phishing pages
  - Trade-off between control and legitimacy/reputation of the domain delivering the attack
  - Attacker domains
    - Name can have a similar spelling, possibly using unicode chars https://www.BankOfThe**VV**est.com (V+V , not W)
    - Or contain the target domain name: http://**paypal.com**.recovery-suspicious.com/
  - Compromised site or free hosting service
    - Domain has better reputation, may be relevant to the attack, may have EV certificates
    - Attacker has less control
    - Phishing page must be hidden out of sight
      - For example, in Let's Encrypt's /.well-known/pki-validation/

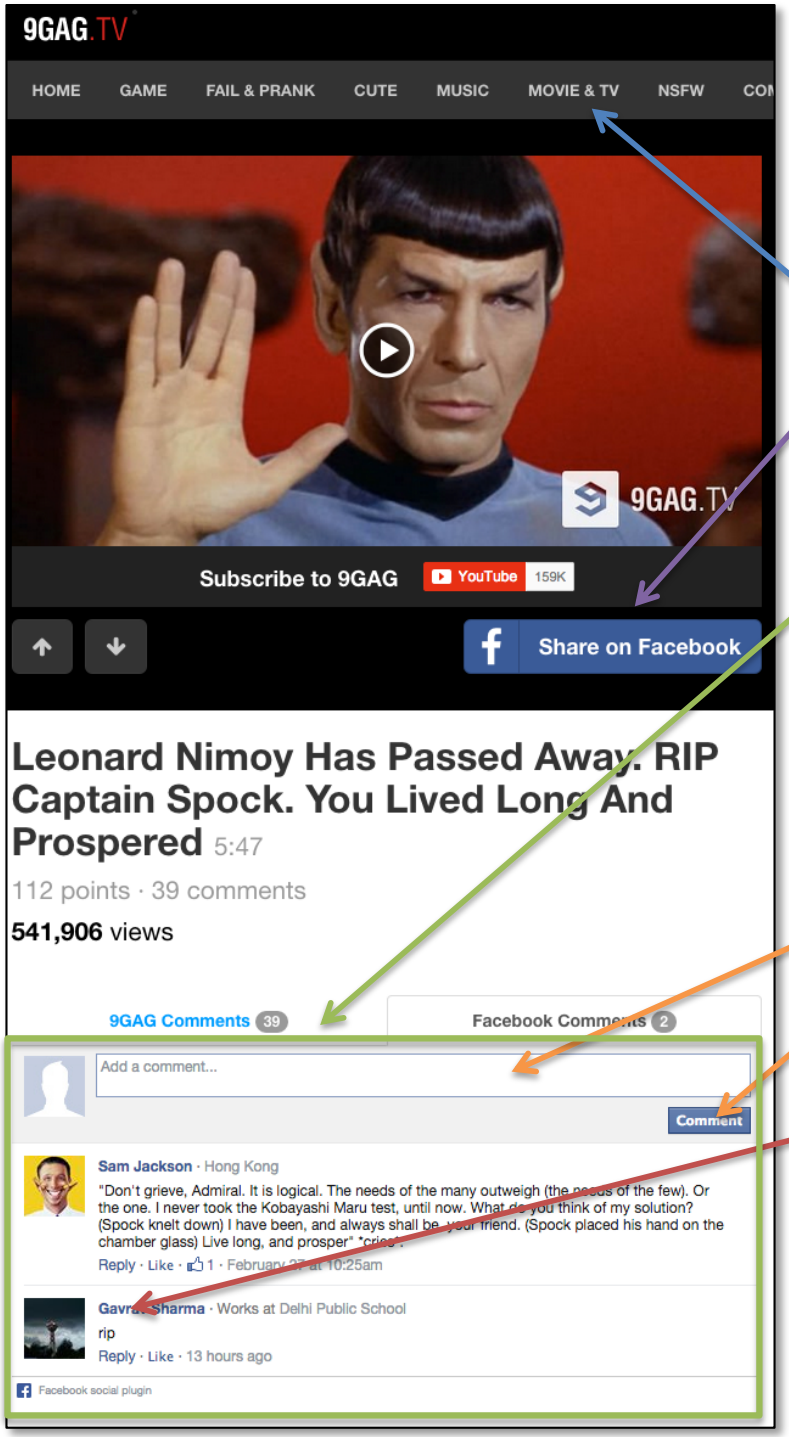
Attacks: phishing

Imperial College London

- Phishing sites are often generated by phishing kits
  - Security researchers found phishing kits accidentally exposed in subdirectories of phishing sites
  - Analysis of code reveals that some kits authors cheated their "customers" by hiding obfuscated PHP code to send themselves copies of stolen data
  - "*There is No Free Phish: An Analysis of "Free" and Live Phishing Kits*" (Cova et al)
- See https://www.phishtank.com for latest reported phishing sites
- Phishing countermeasures
  - Prevent spreading of links via spam
  - Safe-browsing (black)lists of major browsers include also phishing sites
  - Research: automated detection via machine learning
    - Length, complexity of URL
    - Visual or structural similarity of page to whitelisted target page
    - Topological properties of the page
      - Include remote favicon
      - Does not load ads
      - Ratio of internal/external resources

HTML

Imperial College London




```
<HTML>
<head>
…
<link rel="stylesheet"
 href="//cdn-jarvis-ftw.9gaging.com/…/….css”>
<script type="text/javascript" …
 src="//apis.google.com/js/platform.js”></script>
</head>
<body>
…
<iframe id="f15caa161c” …
 title="Facebook Social Plugin” …
 src="https://www.facebook.com/plugins/
      comments.php?api_key=…&amp;width=617”>
<form … method="post"
 action="/ajax/connect/feedback.php" … >
   <input type="hidden" name="lsd"
    value="AVrPxCTM" autocomplete="off”>
   <textarea title="Add a comment..." … >
   </textarea>
   <input value="Comment" type="submit” … >
   …
</form>
<img class="img" width="50px" height="50px"
 src="https://fbcdn-profile-a.akamaihd.net/….jpg”>
…
</iframe>
…
</body>
</HTML>
```

HTML and security

Imperial College London

- Scripts, links, forms and iframes are the key building blocks for (in)secure web applications
  - Considered trustworthy, the subtlety is how to use them correctly
  - Some browsers interpret special cases in bizarre ways
    - See the *Browser Security Handbook*
- HTML5 allows for a variety of other security-sensitive elements
  - Java, Flash, ActiveX, Silverlight, PDF
    - Handled by browser plugins
    - Have their own security restrictions
      - Do not coincide with browser security restrictions
      - Sometimes user-configurable
        - » Storage, camera permissions in Flash
    - Often suffer from memory corruption or other vulns that can lead to browser compromise
  - <audio>, <video>, <canvas> elements
    - Embed rich content directly in HTML page
    - Consistent with HTML5 security model

JavaScript and DOM

Imperial College London

- The Browser Object Model (BOM) provides JavaScript interface to the browser
  - window is the global object (top level variable scope)
  - It defines JavaScript APIs, and in particular
    - Navigator, location, screen, history and document
  - Scripts can
    - Create and navigate windows
    - Access cookies and local storage
    - Manipulate browser history
    - Set timeouts
  - Example: log browser version

```
console.log(navigator.userAgent);
```

- The Document Object Model (DOM) is rooted in window.document
  - Provides JavaScript with an object-oriented interface to the page HTML structure
  - Scripts can
    - Alter the HTML structure of the page
    - Directly read/write data from/to the page
    - Manipulate and submit forms
    - Create and listen to events
  - Example: add a script to the page

```
var x = document.createElement('script');
x.setAttribute('src','http://example.com/script.js');
document.body.appendChild(x);
```

AJAX

Imperial College London

- Asynchronous JavaScript And XML (AJAX)
- Scripts can exchange data with the server without causing the page to reload

```
var x=new XMLHttpRequest();
x.open('GET','http://example.com/data.txt',true);
x.send();
```

  - Asynchronous communication is key to responsive web pages

```
x.onreadystatechange=function(){
  if (x.readyState==4 && x.status==200)
     alert('received: ' + x.responseText);
};
```

- Most commonly data is encoded as JavaScript Object Notation (JSON)
  - More concise than XML
  - Serializes data using JavaScript literal array and object syntax

```
{"student": {
  "name": "Bob",
  "age": 22,
  "grades": ["A+", "B", "B", "A*", "A"]}
}
```

  - Can be parsed simply by using `eval`

```
eval('myJsonObject =' + '{"student": …}');
```

Other HTML5 APIs

Imperial College London

- PostMessasge API: communicate string data between frames
  - Sender frame must be able to obtain a reference to the receiver frame
  - Sender can specify destination origin to prevent eavesdropping
  - Sender code example

```
var dest_iframe = window.frames[3];
var dest_origin = 'https://frame.cybersec.fun:33100';
dest_iframe.postMessage('Hello!',dest_origin);
```

  - Receiver code example

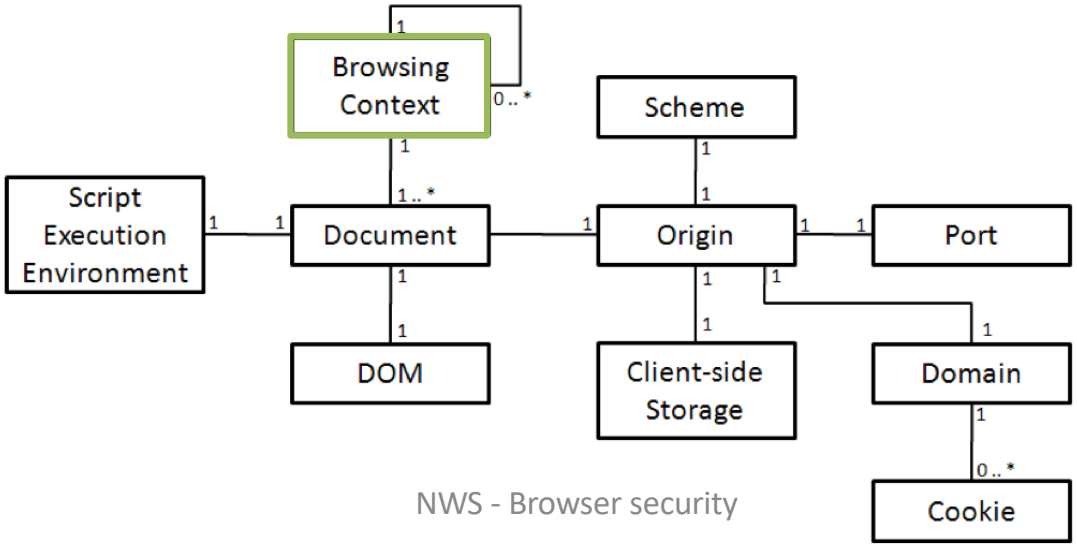
```
var msg_handler = function(event) {
     if(event.origin !== src_origin) return;
     alert('received: ' + event.data);
   }
addEventListener('message',msg_handler,false);
```

- Other HTML5 APIs
  - Web Workers
    - Run batch JavaScript computations in the background, without blocking the page
  - Web Sockets
    - Binary protocol over TCP, to provide bidirectional messaging between client and server
  - WebRTC
    - Real-time communication, for voice/video channels, file sharing
  - Web Cryptography
    - API to provide encryption, signatures, hashing function to JavaScript
  - Other APIs give access to underlying device
    - Geolocation, vibration, …

Navigation

Imperial College London

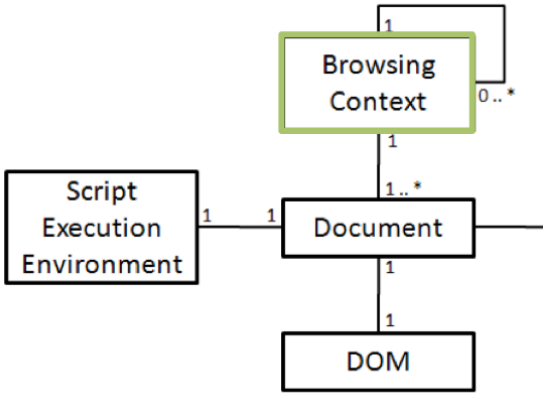
- A *browsing context* (BC) is a container for a web page and its related resources
  - Each browser tab is associated to a top level BC
- BC navigation steps
  - Load a new page (click a link, type in the location bar, script request)
  - Render content
    - Display HTML
    - Execute scripts
    - Fetch and display images and other page resources
    - Navigate nested BCs (frames)
  - Process events
    - User: onclick, onmouseover, …
    - Rendering: onload, onfocus, …
    - JavaScript: timeouts, AJAX responses, postMessage. …



JavaScript in the browser

Imperial College London

- Each browsing context has its document, its DOM and its script execution environment
  - All the scripts embedded in the same document **share the same execution environment** and DOM!
- JavaScript is embedded in a page
  - In the **HTML**
  - In URLs
    - Even within CSS files:
      `div { background-image: url(…); }`
  - In event handlers
- When a script is executed, it can further embed any of the above into the page
- Execution order
  1. Scripts in the <head>, then in the <body>
  2. Handlers of page-loading events: onload, …
  3. Handlers of other asynchronous events
  4. Handlers of page-unloading events: onunload, …

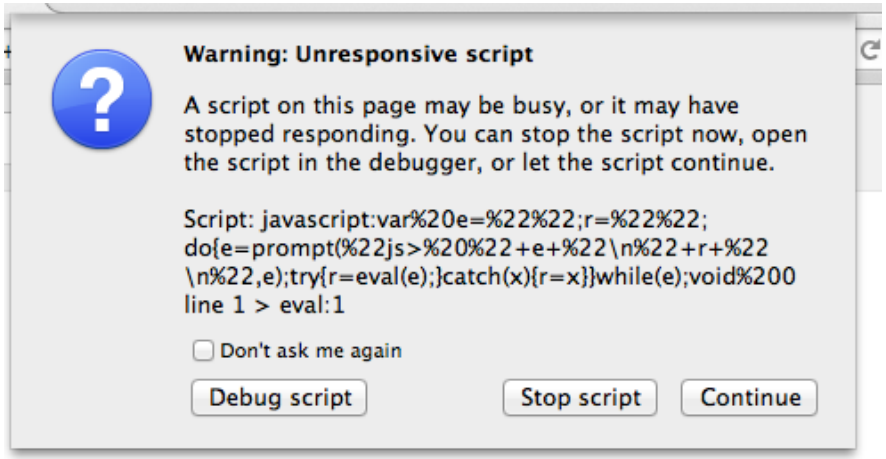



```
<html>
<head>
<script src="http://example.com/script.js" />
</head>
<body>
<div id="id1"> some text </div>
<a href="javascript:f('hello!')">click me</a>
<script>
var f=function(x){alert(x);}
document.getElementById("id1").innerHTML =
        "<h1 onclick='f(0)'>click me</h1>";
</script>
</body>
</html>
```

Loading and executing scripts

Imperial College London

- Each script is executed until its end, without interruptions
  - Unresponsive scripts spoil user experience
    - Browser may open a dialog offering to terminate script
    - Or just hang the tab
  - To keep page responsive
    - Use event handlers, AJAX
    - Outsource computation to same-origin iframe
    - Web Workers: designed to do background JavaScript processing
- Uncatchable exceptions terminate current script
  - Execution moves on to the next script

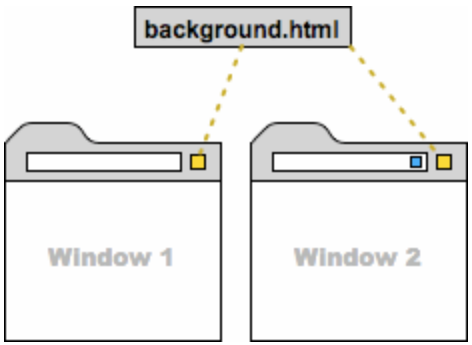


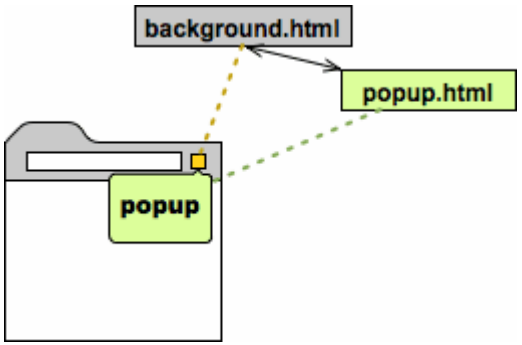

```
<body>

<script>
throw “uncaught exception!”;
alert(“will not happen”);
</script>

<script>alert(“will happen”);</script>

</body>
```

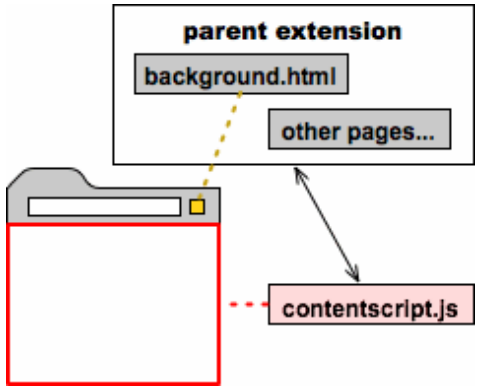
Extensions

Imperial College London

- Plugins extend a browser by adding binary components
  - Run Java, Flash, and other applications
- Extensions are JavaScript-based, and talk to the browser API
  - Developer toolbars, password managers, NoScript, …
- Main extension components
  - Extension icon in the chrome
  - Background page: the extension "back end", always on
  - Other extension pages: icon's popup window, possibly regular browser windows
  - Content scripts: interact with the DOM of each visited page
    - Hence, can also add scripts in the execution environment of visited page
    - Need postMessage to talk to other extension pages







Attacks: clickjacking

Imperial College London

- User communicates data and intentions to the browser via clicks & keystrokes
  - Attacker tries to interfere with “agreement” between user and browser
- Example: overlay transparent Twitter iframe on attractive website
  - User thinks she clicks on PLAY! button but she’s deleting her Twitter account

- Countermeasures
  - X-Frame-Options header
    - `X-Frame-Options: DENY`
      - Page in response body cannot be contained in iframe
    - `X-Frame-Options: SAMEORIGIN`
      - Page can be iframed only by a page in same origin
    - `X-Frame-Options: ALLOW-FROM https://example.com`
      - Page can be iframed only by a page in designated origin
      - This option is not as widely supported
  - Frame-busting JavaScript code can try to prevent page from being iframed

```
if (self !== top) top.location = self.location;
```

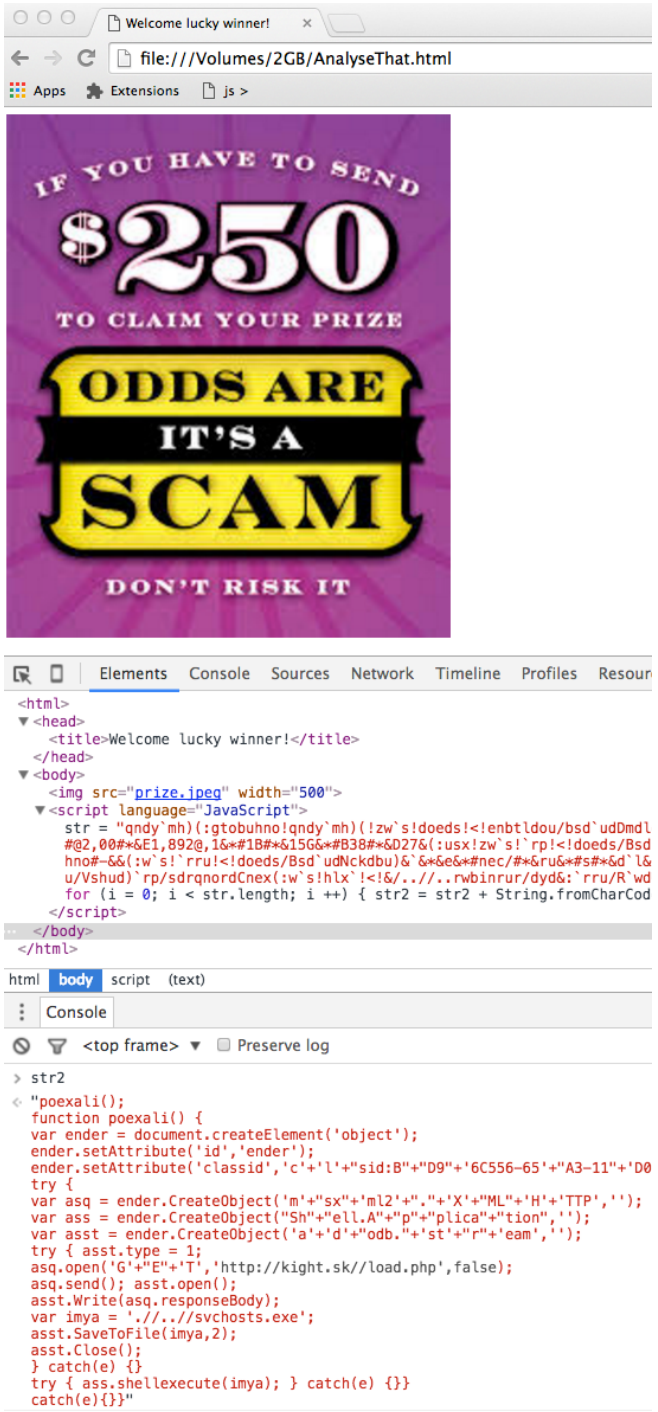
Attacks: drive-by download

Imperial College London

- User visits malicious page
- Page exploits vulnerability
  - Browser memory corruption (stack/heap overflow, …)
  - In plugins: Java, ActiveX, Flash, PDF…
  - In browser extensions (new!)
- Exploit installs malware on client machine
  - Or at least saves dangerous file that can be accidentally opened
- Concrete example on next slide
- Countermeasures
  - Disable running plugins in the browser, or confine them to a restricted sandbox
  - Warn user visiting blacklisted websites (Google Safe Browsing, etc.)
  - Detect suspicious JavaScript with an IDS
  - Harden browser and OS against memory corruption vulnerabilities
    - Chrome-like architecture to isolate compromised processes
    - ASLR and other low-level defenses
    - Spectre/Meltdown exploitable via the browser using JavaScript

Dissecting a drive-by

- Find source code and analyse its structure
  - Likely to involve obfuscation to avoid detection
  - Code may need to deobfuscate itself at some stage
  - Inspect variables at intermediate steps of the execution to save analysis effort
- Identify the real exploit code
  - Calls plugin with a suspicious payload?
    - Try to identify plugin, payload
  - Builds large DOM structures or JavaScript strings?
    - May be trying to exploit browser memory corruption
  - Look for a corresponding CVE
    - If you can't find it, you may have a 0-day!
- Tools for the trade
  - Developer toolbar, web proxy, encoding libraries
- Our example
  - Calls ActiveX object
    - clsid:BD96C556-65A3-11D0-983A-00C04FC29E36
  - Google search reveals
    - Microsoft Exploit: HTML/MS06014



Content sniffing and polyglots

Imperial College London

- Content sniffing
  - A browser may render a resource based on the resource data instead of the MIME type reported in the Content-Type header
  - This allows pages with broken MIME types to still work correctly
- Polyglots
  - File that are valid with respect to different data formats
  - HTML files are valid text files, and viceversa
  - GIFs, PNGs, JARs have been crafted to be valid JavaScript or HTML
- Security risks
  - Polyglots can bypass content-based filtering rules
  - Polyglots + content sniffing can lead to bypassing the SOP
    - `http://example.com/upload.php` accepts any file as an image
    - Server checks that uploaded file is a valid image before serving it in image gallery at `http://example.com/images/`
    - Attacker uploads PNG/HTML polyglot with filename `attack.html`
    - Attacker sends victim a link to `http://example.com/images/attack.html`
    - Attacker now controls code in `http://example.com` origin
- X-Content-Type-Options
  - Response header to prevent content sniffing
  - `X-Content-Type-Options: nosniff`