Mitigating Malware

Collin Jackson

CS142 – Winter 2009

Approach

- Fact: Browsers will always have bugs
- Goal: Reduce the harm

Frequency of interactions with attacker × Percentage of time vulnerability is unpatched × Damage if attack works

= Harm

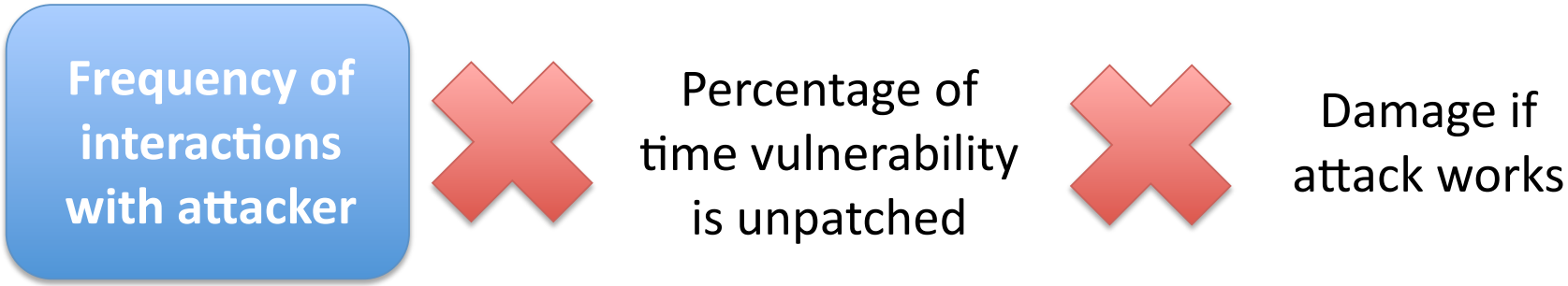
Outline

Frequency of interactions with attacker

Percentage of time vulnerability is unpatched

Damage if attack works

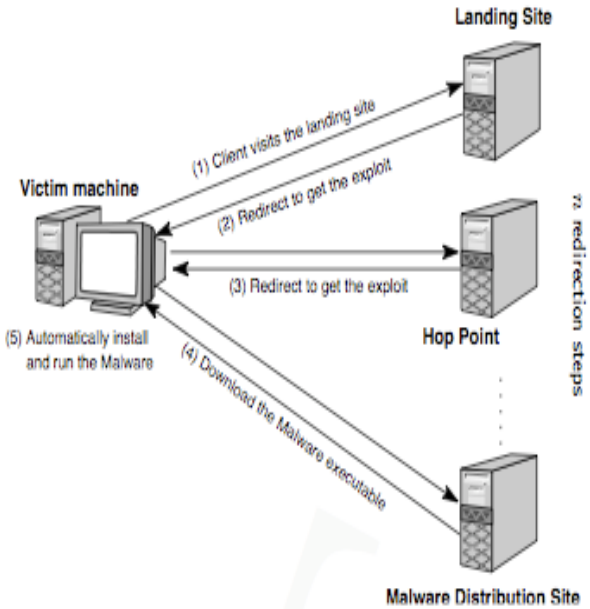
1. Preventing the Introduction
2. Vulnerability Response
3. Failure Containment



PREVENTING THE INTRODUCTION

Drive-by downloads

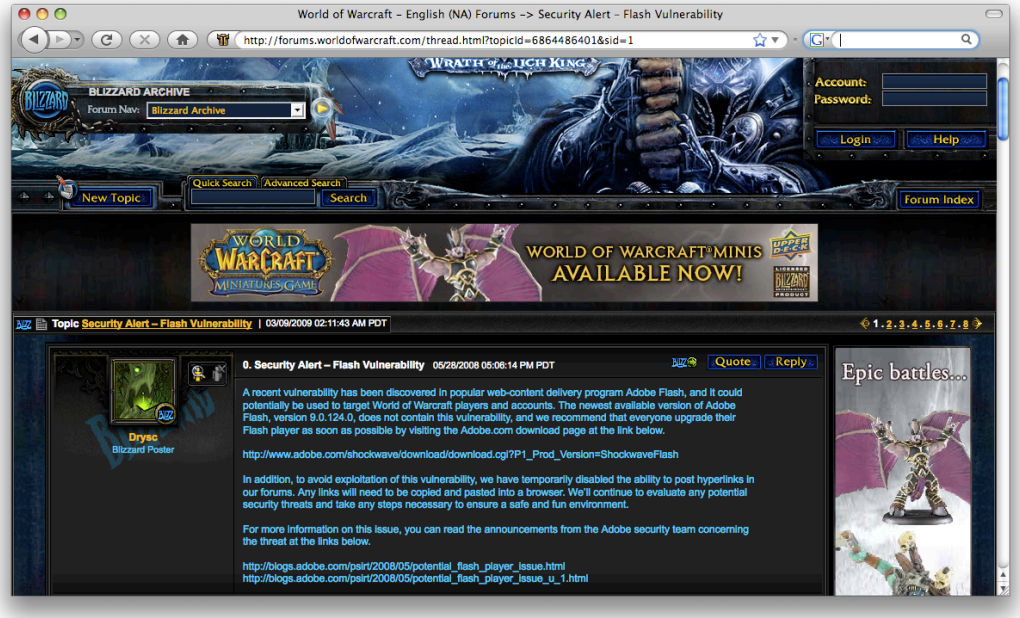
- Silently installs software when web page is loaded
- Increase exposure by compromising other sites and insert code into them
- Sites owners unaware they are participating in an attack



Provos et al. "All your iFRAMES Point to Us"

World of Warcraft keylogger

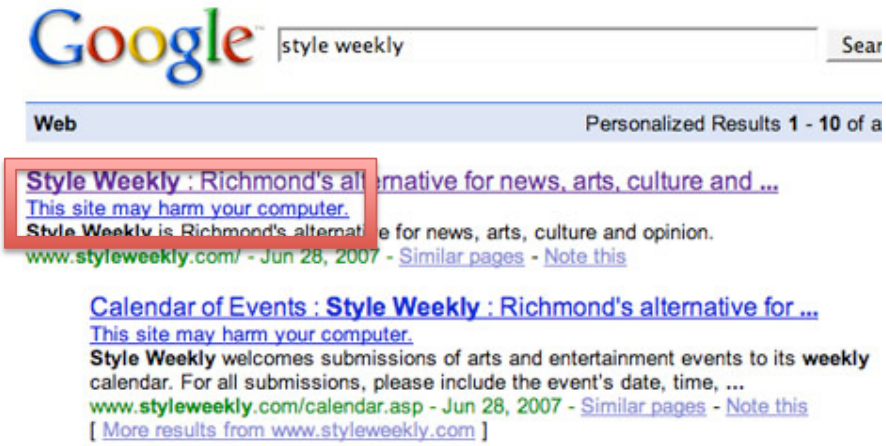
- Flash Player exploit used to install keylogger
- Links to malicious SWF posted on forums



- "Solution": Disable hyperlinks on forum

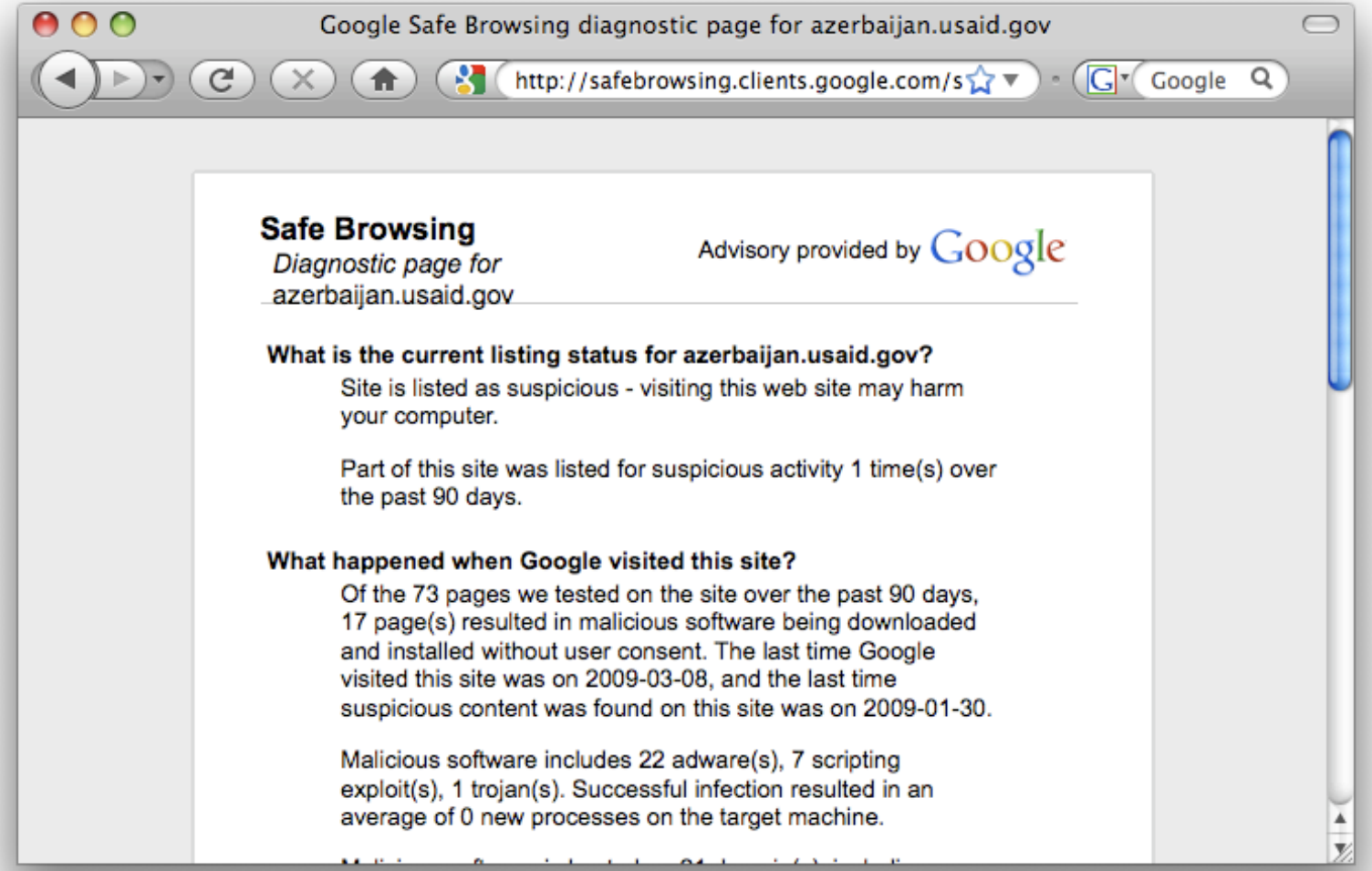
Scaling it up to the entire web

- 1.3% of the incoming search queries to Google returned at a least one malware site
- Visit sites with an army of browsers in VMs, check for changes to local system
- Indicate potentially harmful sites in search results



Now do it in the browser

Helping the webmaster out

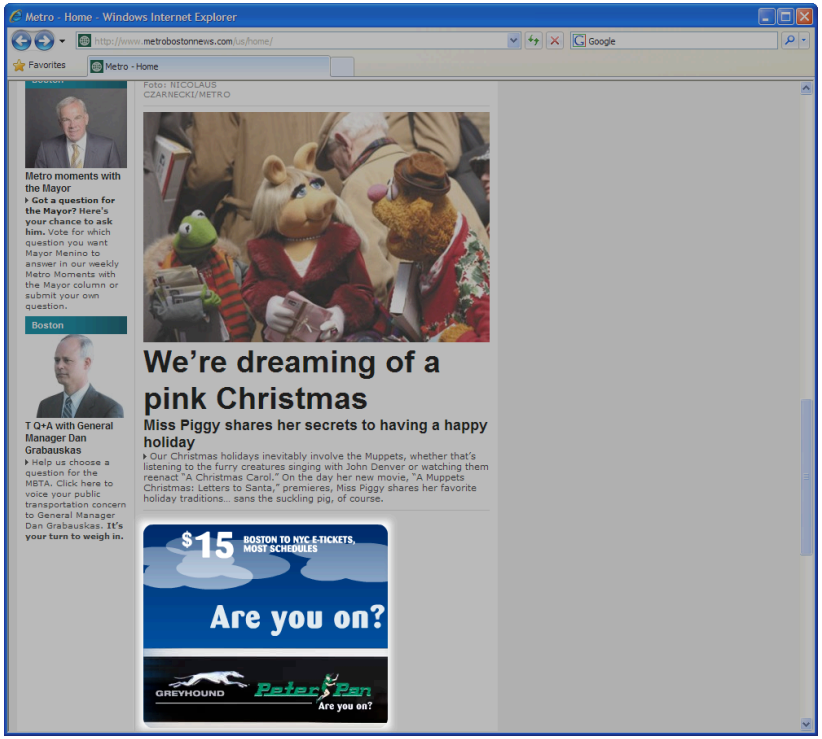


Introductions are easy

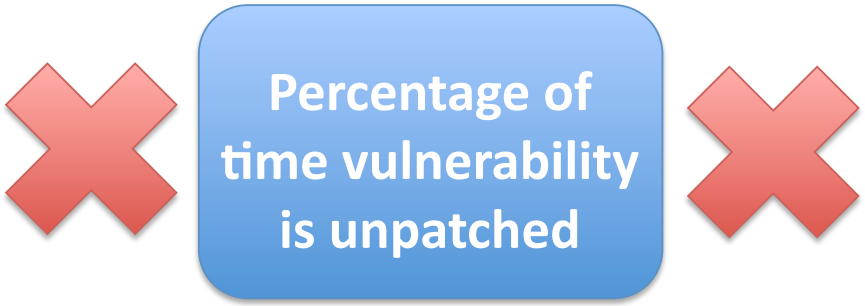
- Impressions are cheap ($1 = 2000)
- Ad that is harmless today may be malicious tomorrow
- Possible mitigations:

  `<iframe security=restricted>`

  `<iframe sandbox>`



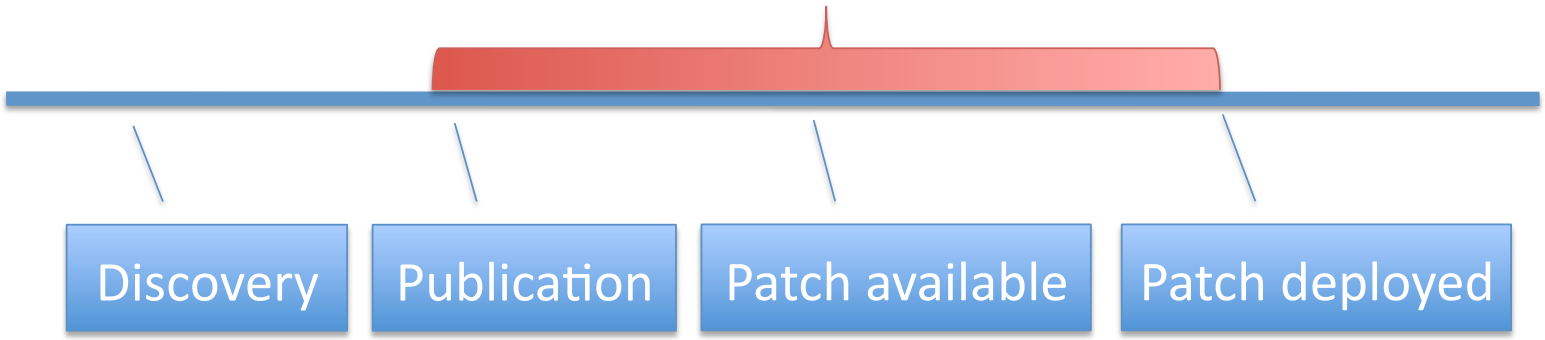
Frequency of interactions with attacker



Damage if attack works

VULNERABILITY RESPONSE

Closing the vulnerability window

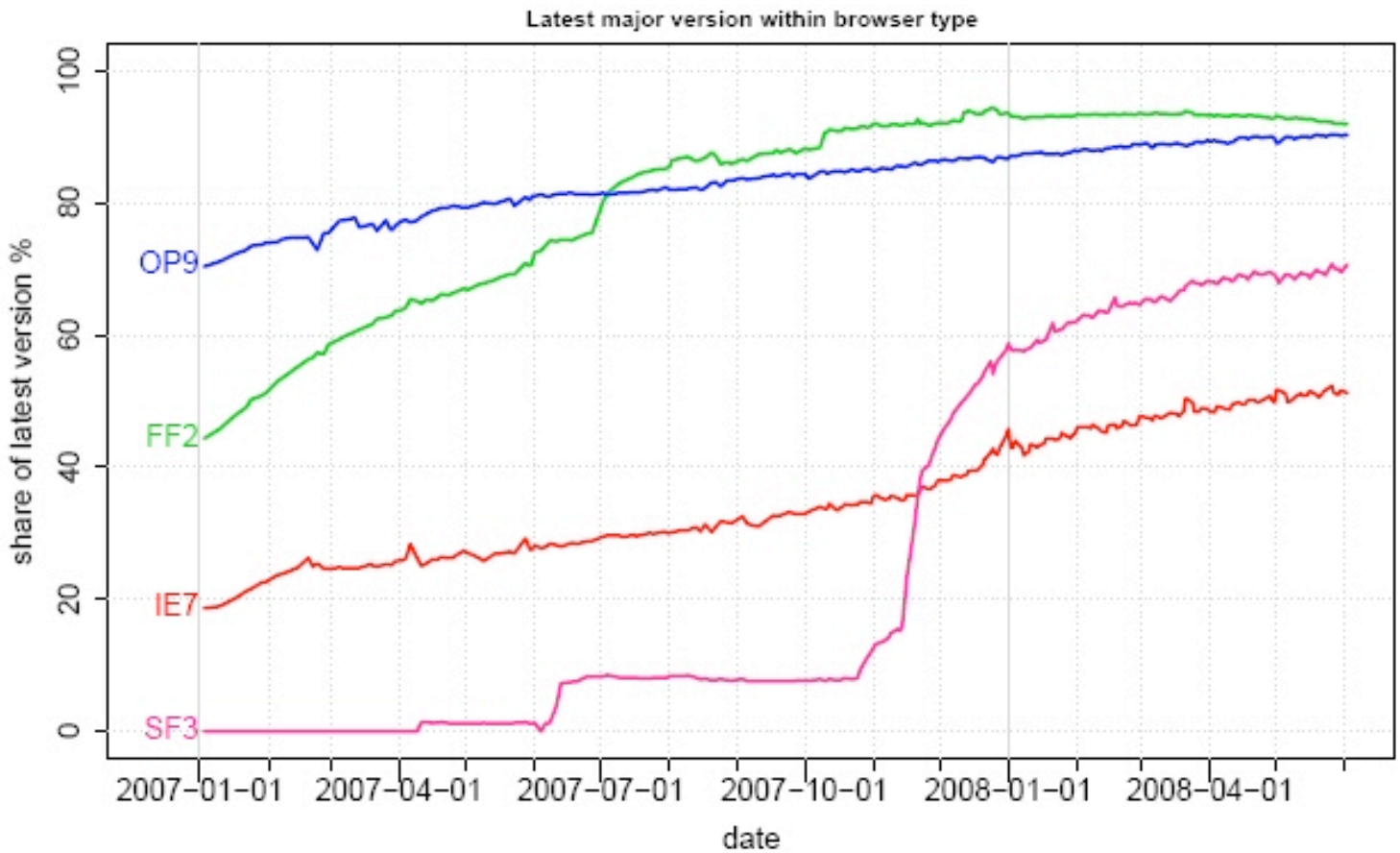


- Delay publication
  - Coordinate with security researchers
  - Offer prizes for responsibly disclosed security bugs
- Make patch available faster
- Deploy patch faster

Obstacles to patch deployment

- Interrupts work flow
- Requires adminstrator privileges
- Risk of breaking things
- Separate update mechanisms
- Silent approach: GoogleUpdate.exe

Getting better, but not fast enough



Frei et al. Examination of vulnerable online Web browser populations and the "insecurity iceberg"

Announcements

- Office hours have moved

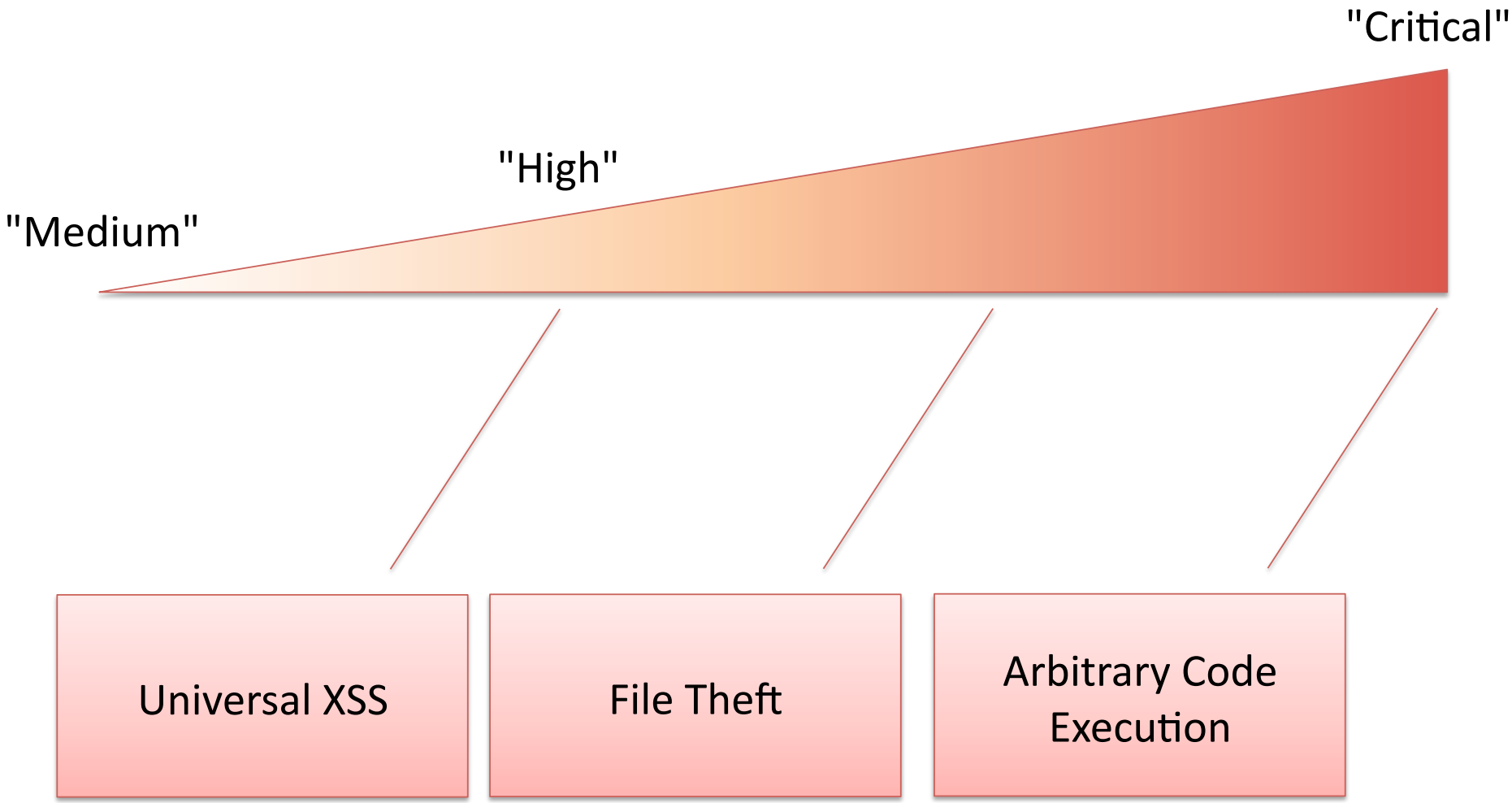
Frequency of interactions with attacker

Percentage of time vulnerability is unpatched

Damage if attack works

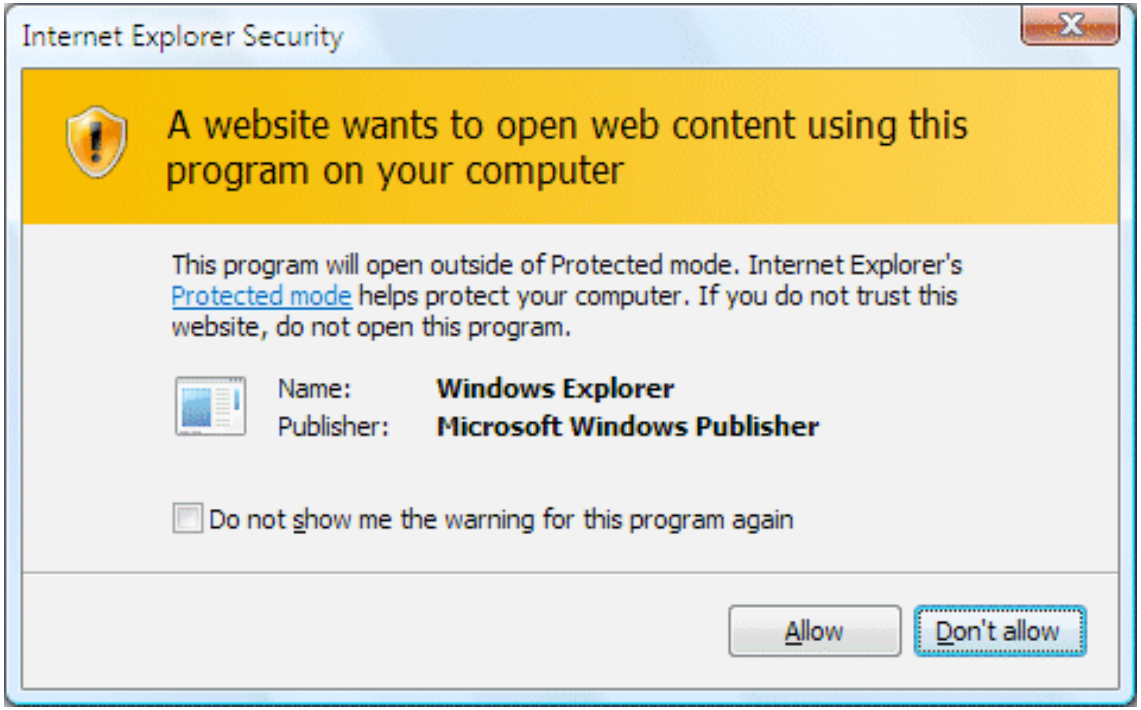
FAILURE CONTAINMENT

Severity


"Critical"
"High"
"Medium"
Universal XSS
File Theft
Arbitrary Code Execution

Protected Mode IE

- IE7 in Vista is a "low rights" process
- Can prompt user to get more privileges

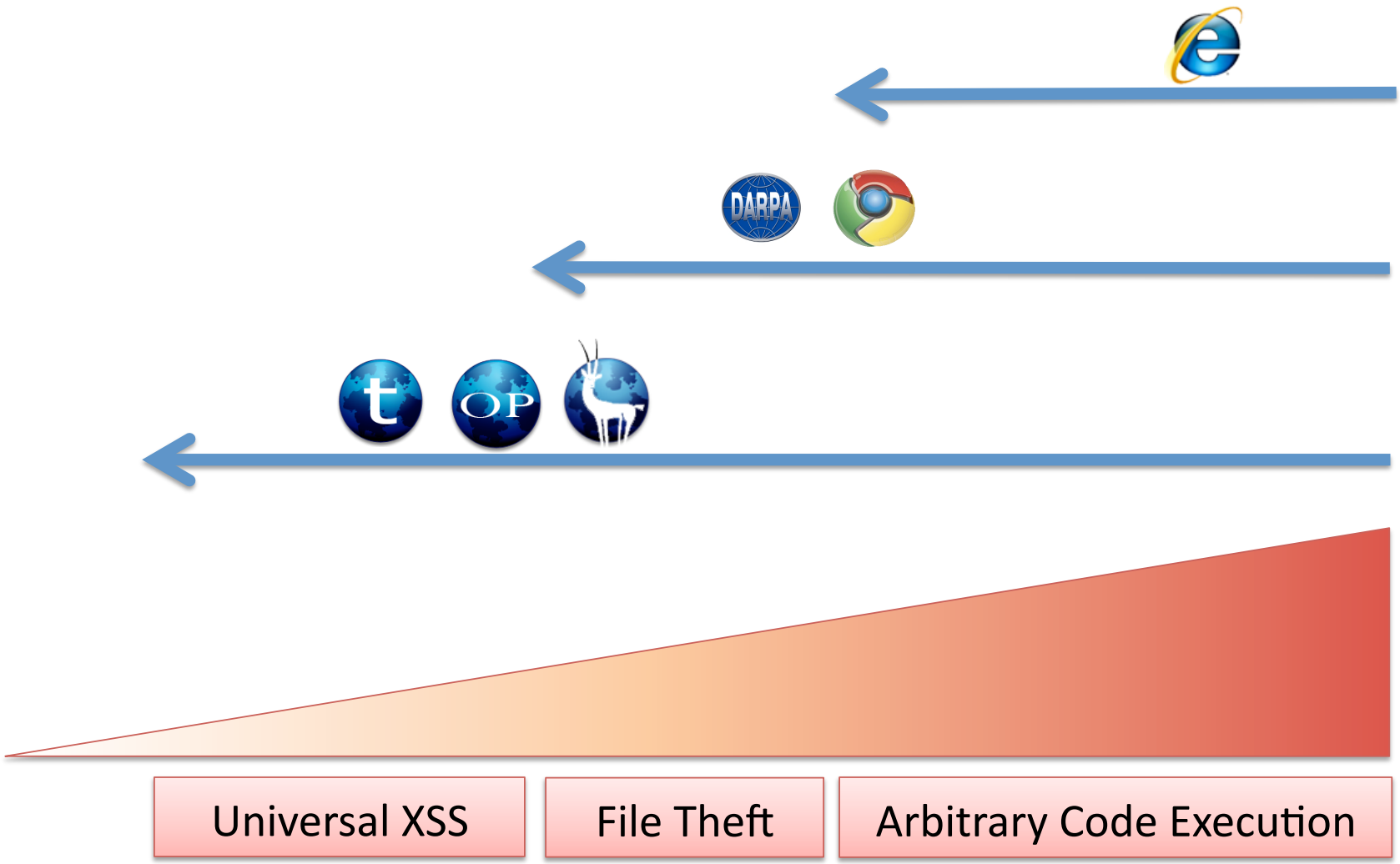


IE7 Containment Goals

- Arbitrary code execution won't let attacker:
  - Install software
  - Copy files to startup folder
  - Change homepage or search provider setting

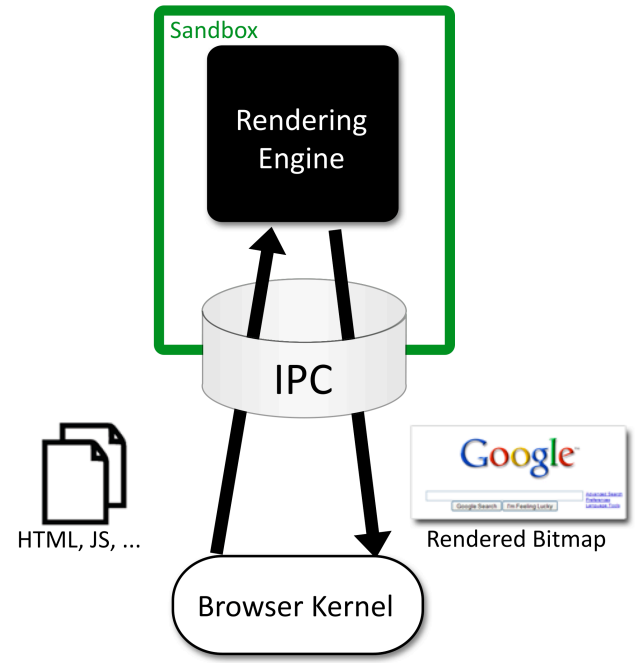
- Can we do more?

Containment Goals



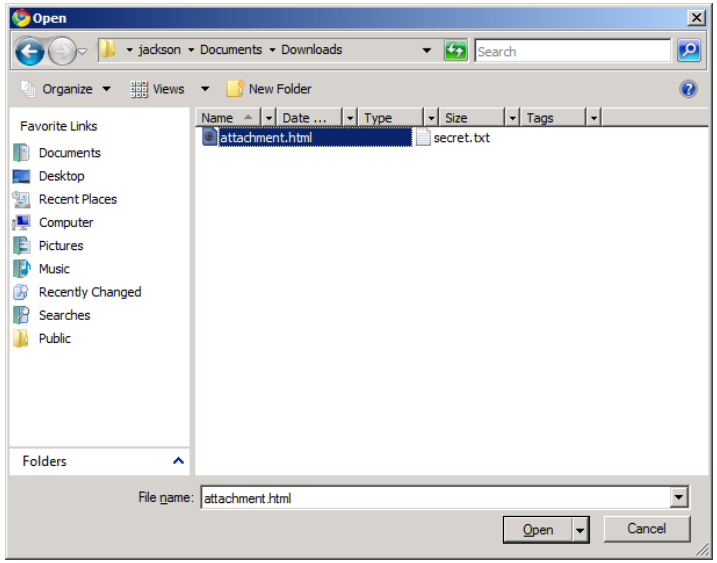
Chromium Security Architecture

- Browser ("kernel")
  - Full privileges (file system, networking)
  - Coarse-grained security policies protect local system
- Rendering engine
  - Sandboxed
  - Fine-grained same origin policy enforcement
- One process per plugin
  - Sandboxing optional



Barth et al. "The Security Architecture of the Chromium Browser"

Preventing File Theft



- File Downloads.
  - Renderer can only write files to My Documents\Downloads
- File Uploads.
  - Renderer is granted ability to upload file using browser kernel's file picker.
- Network Requests.
  - Can only request web-safe schemes (http, https, ftp)
  - Dedicated renderers for file://

Task Allocation

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

Is the "kernel" too complex?

- Total CVEs:

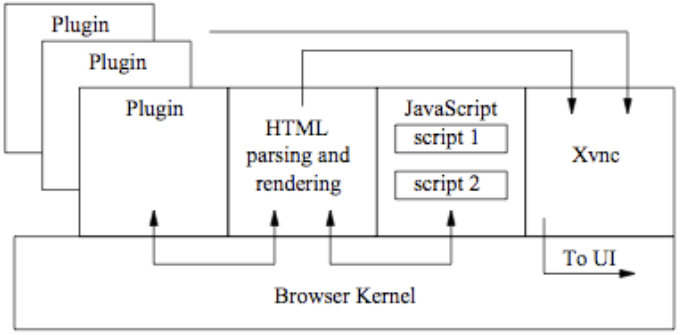
	Browser	Renderer	Unclassified
Internet Explorer	4	10	5
Firefox	17	40	3
Safari	12	37	1

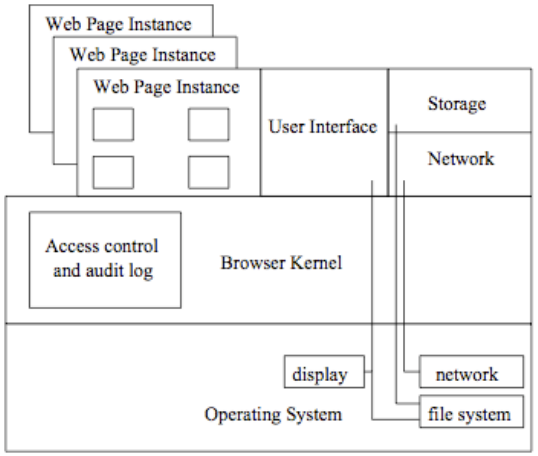
- Arbitrary code execution vulnerabilities:

	Browser	Renderer	Unclassified
Internet Explorer	1	9	5
Firefox	5	19	0
Safari	5	10	0

OP Browser

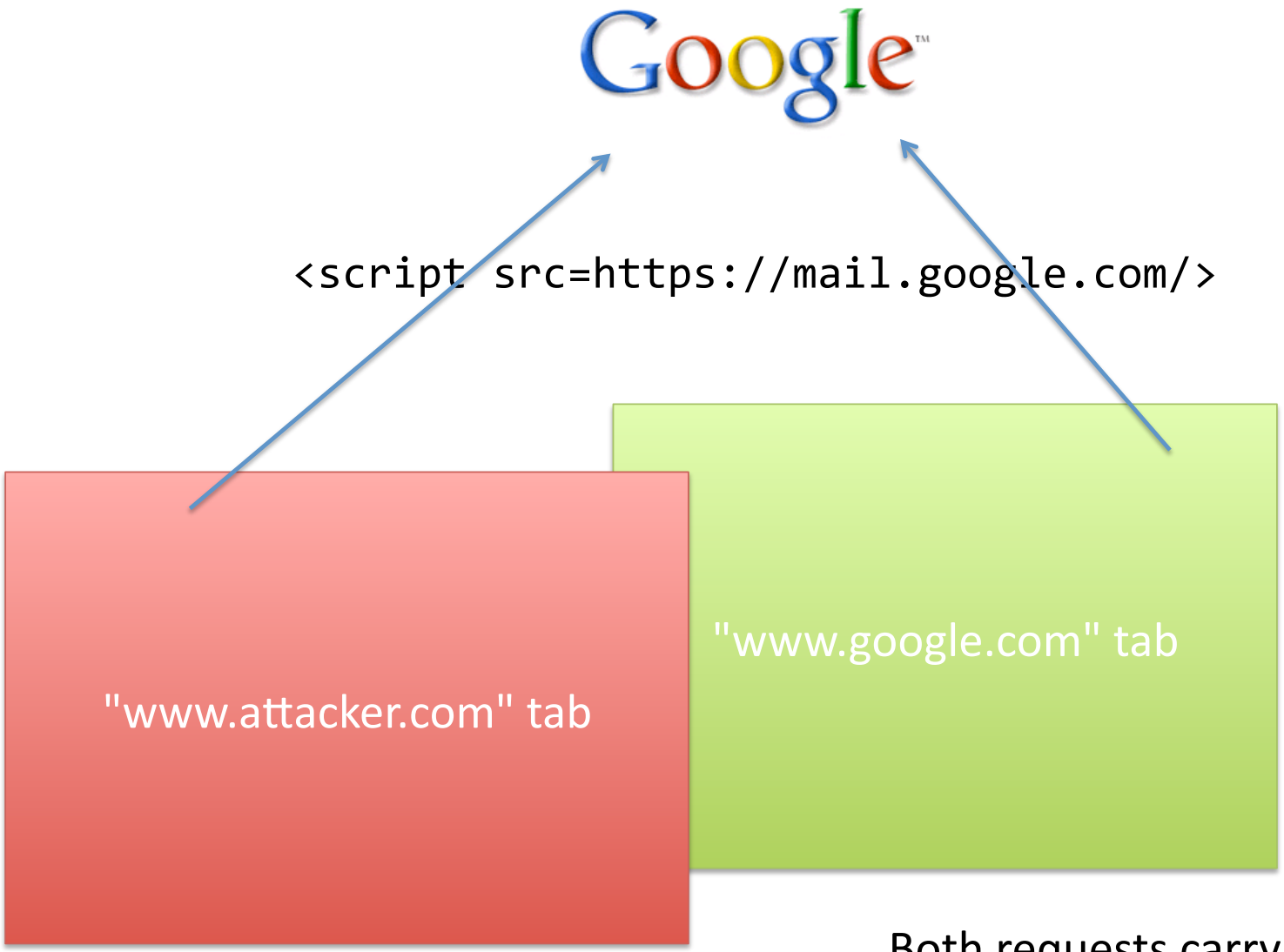
- Fine-grained componentization
- Want to mitigate UXSS
- Focus is on plugin containment
  - Will plugins refuse to be contained?
  - Historically a platform for innovation in policy
- Missing a basic issue…





Grier et al. "Secure web browsing with the OP web browser"

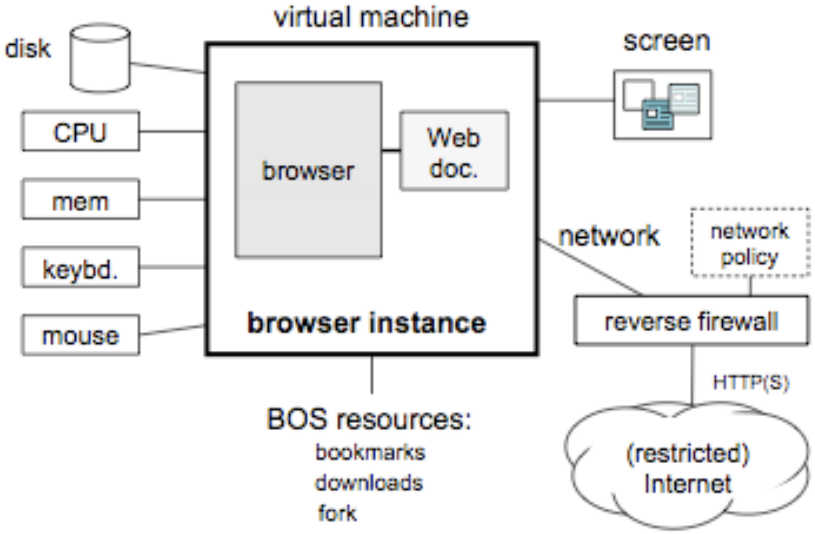
Why UXSS Containment is Hard



Both requests carry cookies!

Tahoma's Approach

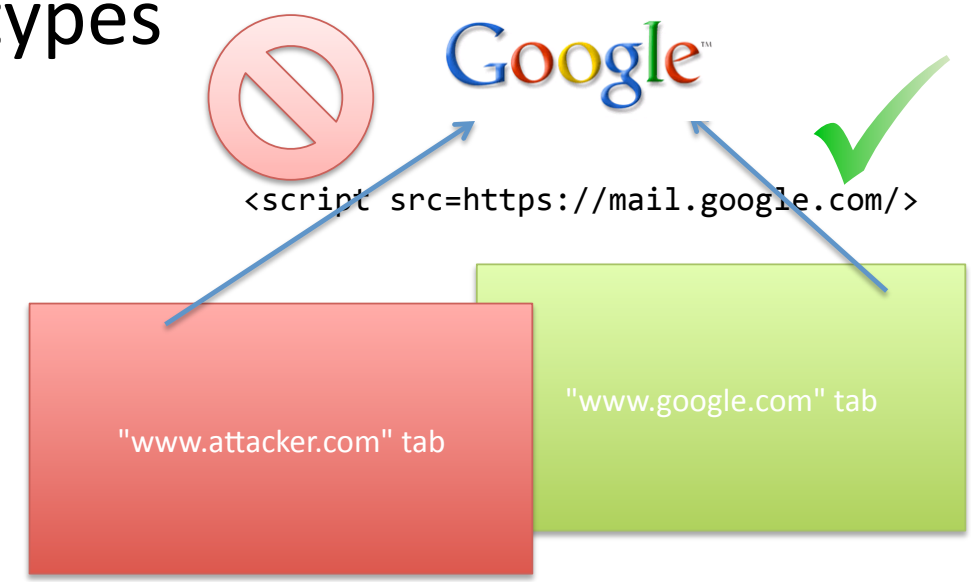
- Very coarse grained policy
- Separate browser state for each top-level site
- Site can opt in to more sharing via manifest files

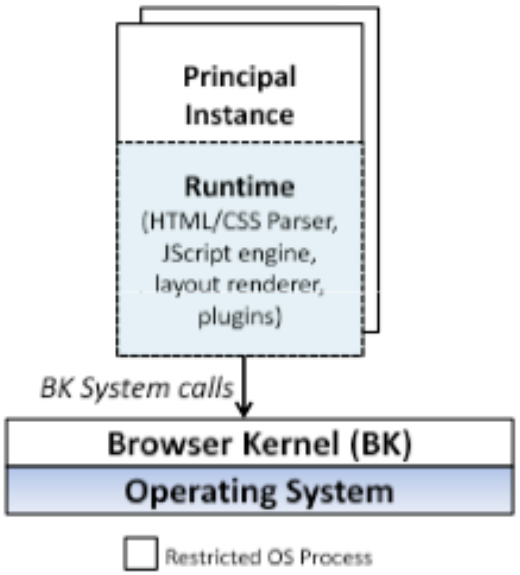


Cox et al. "A Safety-Oriented Platform for Web Applications"

Gazelle's Approach

- Inspect cross-origin HTTP responses
- Filter unexpected content types

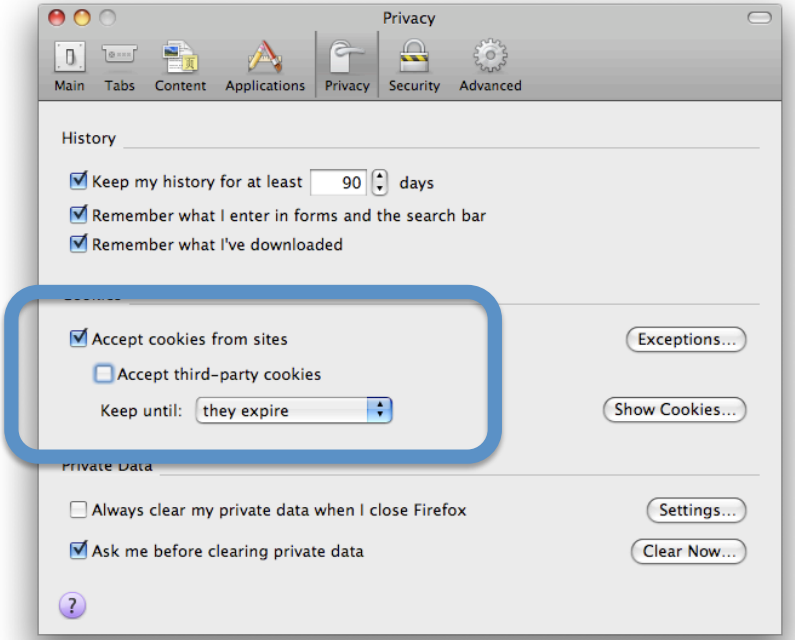




Wang et al. "The Multi-Principal OS Construction of the Gazelle Web Browser"

Another approach: Cookie Blocking

- Block the "Cookie" header for cross-domain resource loads
- Third-party cookie blocking already does this for privacy
- Third-party frames are ok
- Cross-subdomain might be ok



Open question: How many sites does this break compared to content type filtering?

Conclusion

Frequency of interactions with attacker

Percentage of time vulnerability is unpatched

Damage if attack works

1. Preventing the Introduction

2. Vulnerability Response

3. Failure Containment

Reading

- Barth et al. "The Security Architecture of the Chromium Browser"
- Optional (i.e. not required):
  - Provos et al. "All your `iFRAMES` Point to Us"
  - Frei et al. Examination of vulnerable online Web browser populations and the "insecurity iceberg"
  - Cox et al. "A Safety-Oriented Platform for Web Applications"
  - Grier et al. "Secure web browsing with the OP web browser"
  - Wang et al. "The Multi-Principal OS Construction of the Gazelle Web Browser"