The Multi-Principal OS Construction of the Gazelle Web Browser

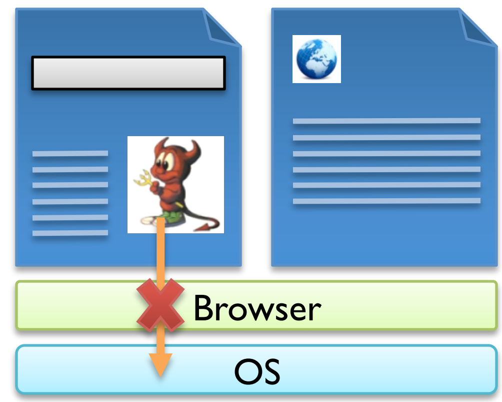
Helen J. Wang, Chris Grier, Alex Moshchuk,
Sam King, Piali Choudhury, Herman Venter

Browser as an application platform

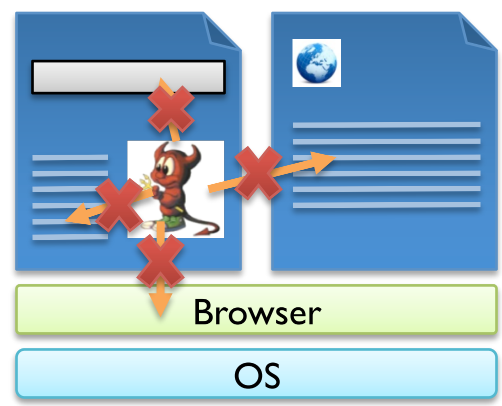
- Single stop for many computing needs
  - banking, shopping, office tasks, social networks, entertainment
- Static document browsing → rich programs
  - obtained from mutually distrusting origins
  - same-origin policy: a browser is a multi-principal platform where *web sites are principals*
- Browser = prime target of today's attackers

Your valuables are online!

- Existing browser security mentality:
  - **valuables on local machine**
  - protect local machine from the web



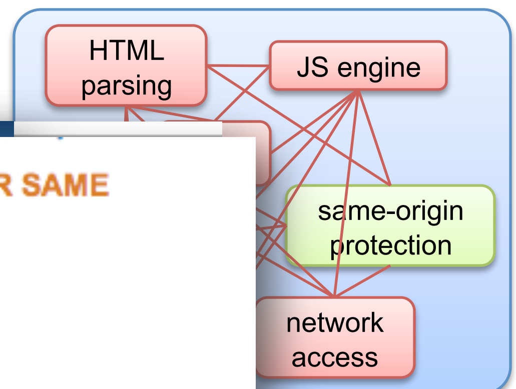
- This work's mentality:
  - **valuables online**
  - must also *protect web site principals* from one another

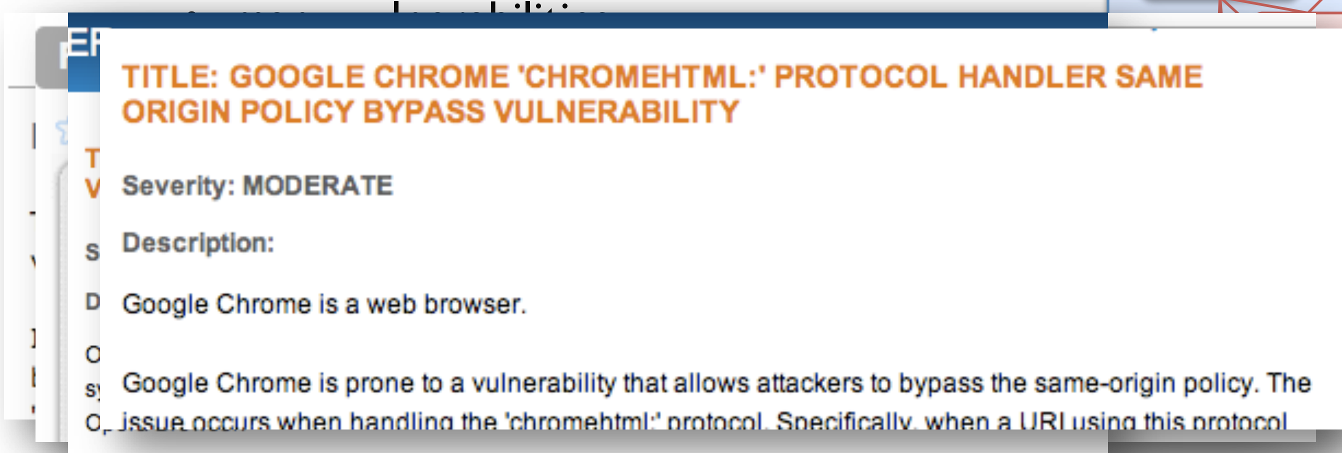


Browser design requires OS thinking

- Cross-principal protection is an essential function of an operating system

- Fundamental flaw with existing browser designs:
  - OS logic is intermingled with application-specific content processing
  - consequences:
    - unreliable cross-principal protection




TITLE: GOOGLE CHROME 'CHROMEHTML:' PROTOCOL HANDLER SAME ORIGIN POLICY BYPASS VULNERABILITY

Severity: MODERATE

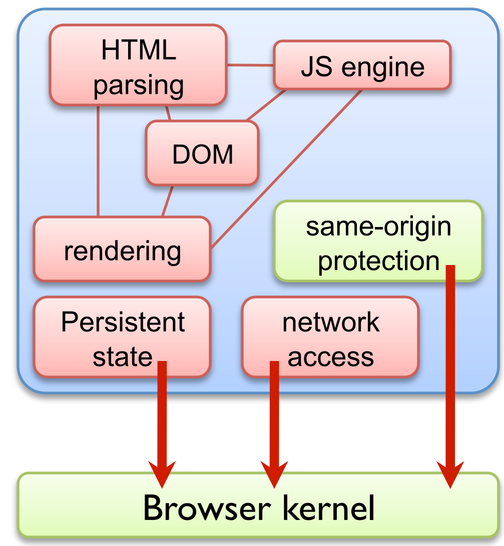
Description:

Google Chrome is a web browser.

Google Chrome is prone to a vulnerability that allows attackers to bypass the same-origin policy. The issue occurs when handling the 'chromehtml:' protocol. Specifically, when a URL using this protocol

Gazelle

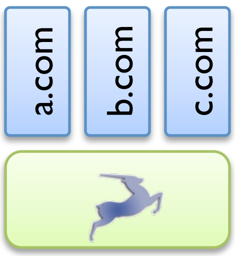
- An OS *exclusively* manages:
  - protection across principals
  - resource allocation
  - resource access control



- Our approach for designing Gazelle:
  - take *all* OS functionality out of content processing logic
  - put it into a small, simple browser kernel

Gazelle

- Build the browser as a multi-principal OS
  - label principals according to web site origins
  - enforce strong isolation among principals
    - apply to *all* resources
    - apply to *all* types of web content



- Challenge: correctly handle web's embedding model (cross-origin mashups)
  - implications for managing display and other resources

Outline

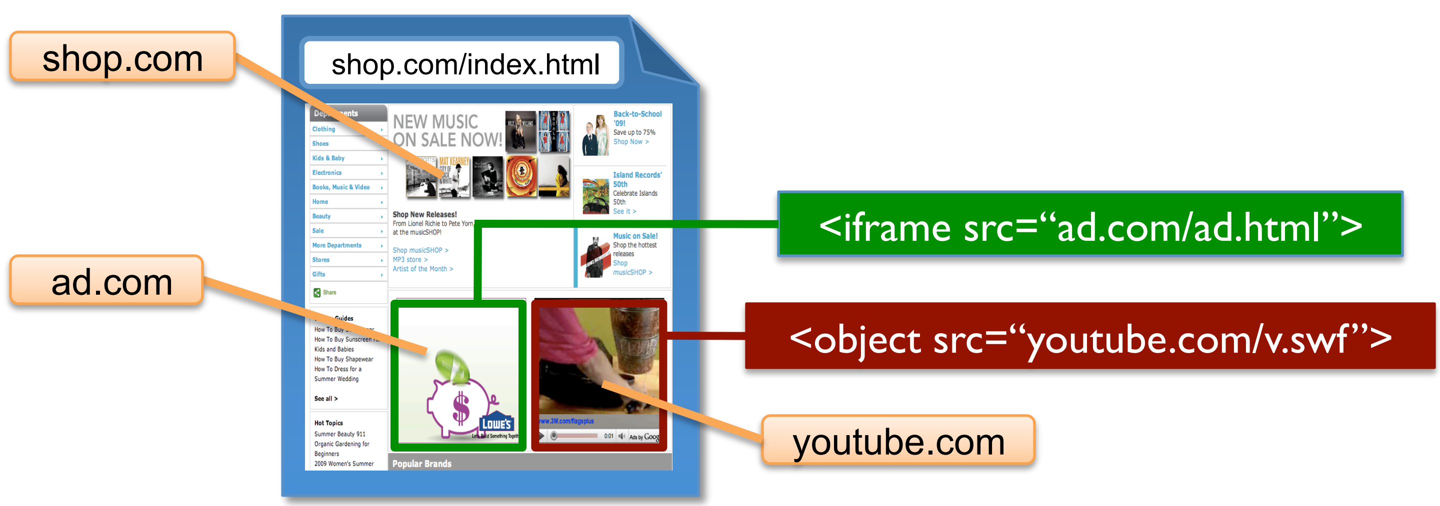
- Motivation
- Gazelle's design
  - defining the principals
  - protection architecture
  - securing the display
- Implementation
- Evaluation

Defining the principals

- Gazelle's principal corresponds to a web site origin
  - origin = <protocol, domain, port>: http://www.news.com:80
- Principal = unit of protection

Labeling embedded content

- <scripts>, stylesheets
  - a library abstraction
  - runs as includer
- <iframe>, <frame>, <object>, <img>
  - runs as provider
  - *same* principal definition for plug-in content



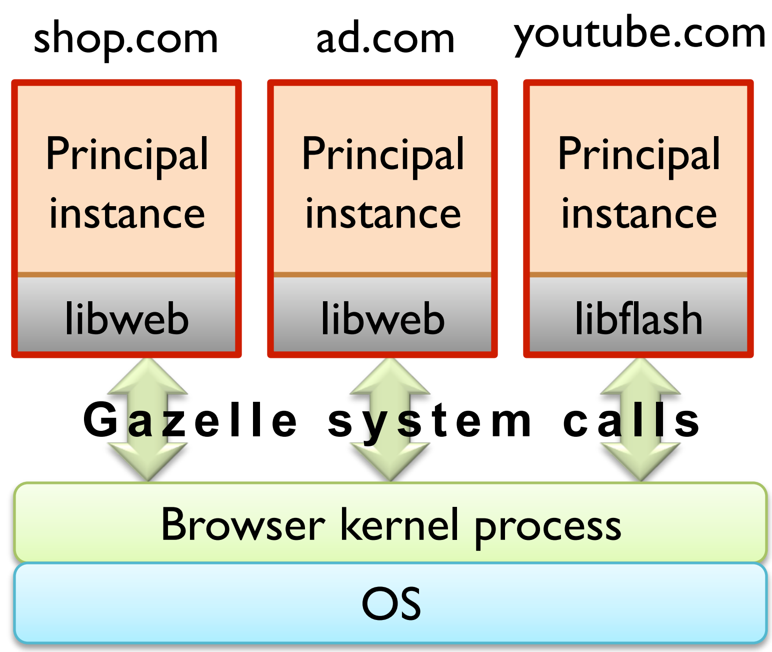
Principal instances

- Can have multiple instances of same principal
- A *principal instance* is:
  - the unit of failure containment
  - the unit of resource allocation
- Principal instances of the same principal share persistent state

Tab 1

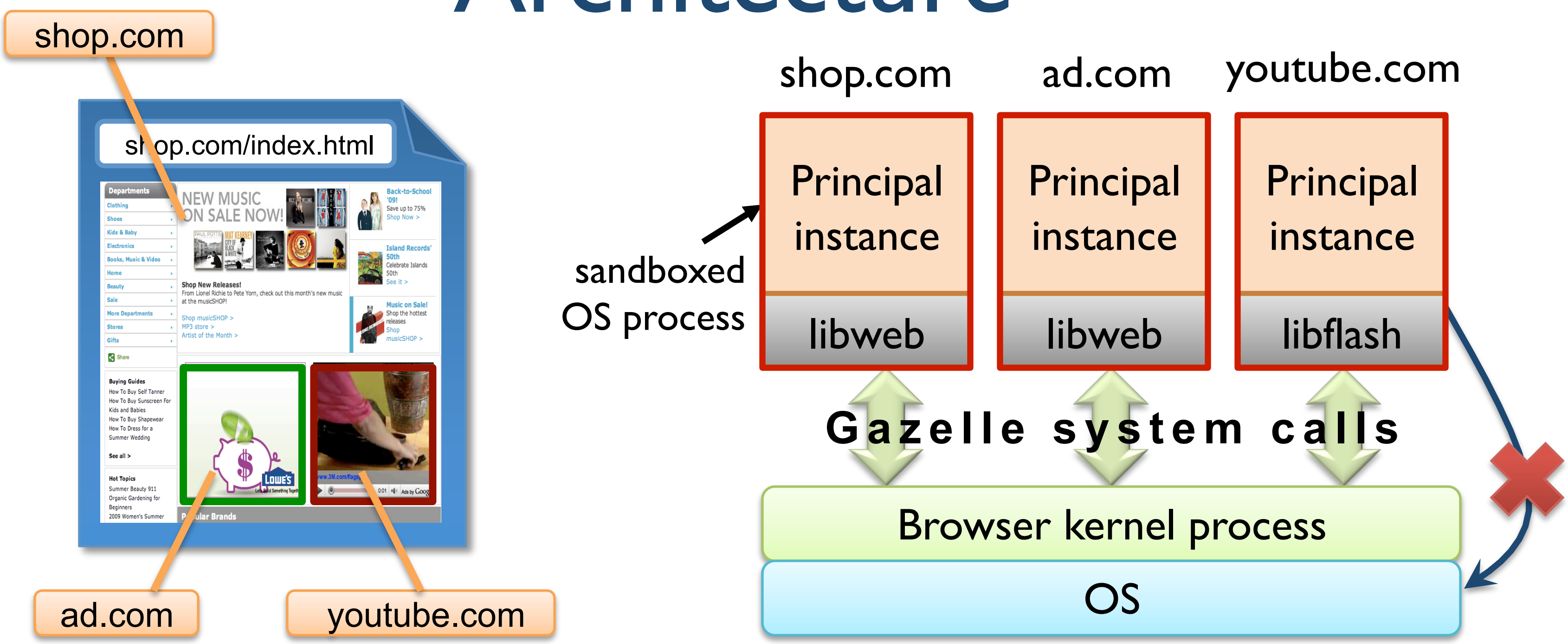
Tab 2

Architecture



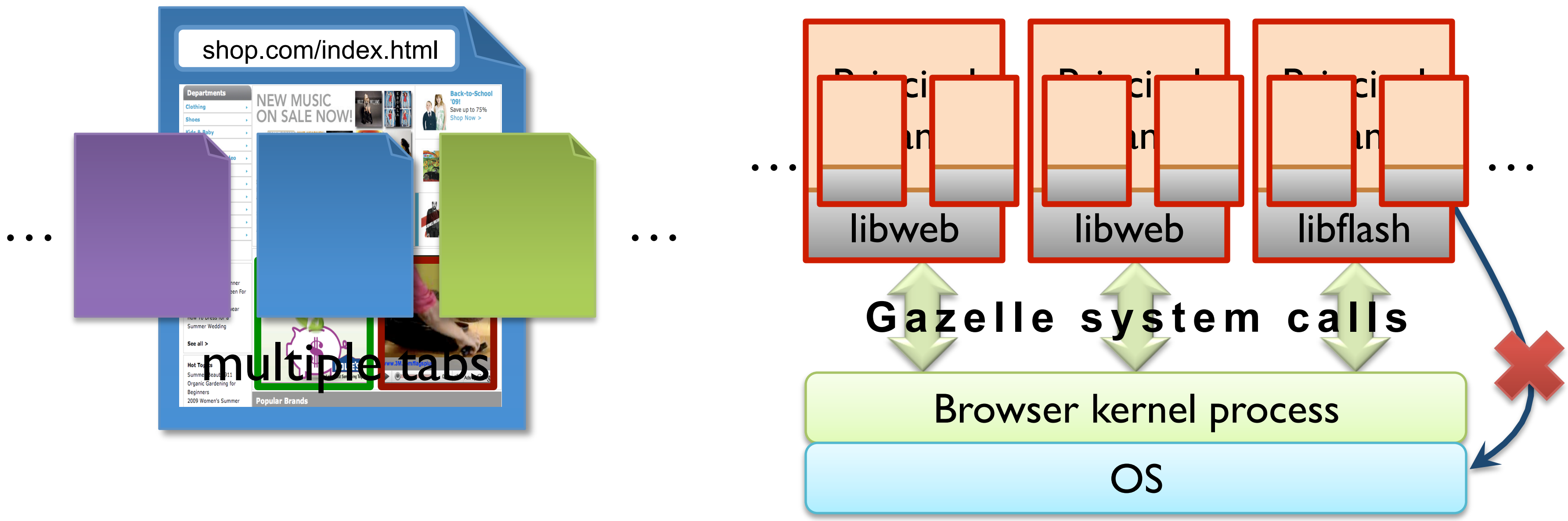
- Browser kernel:
  - *exclusively* manages principals and all system resources
    - processes, network, storage, display, IPC, system events
  - enforces *all* security policies

Architecture



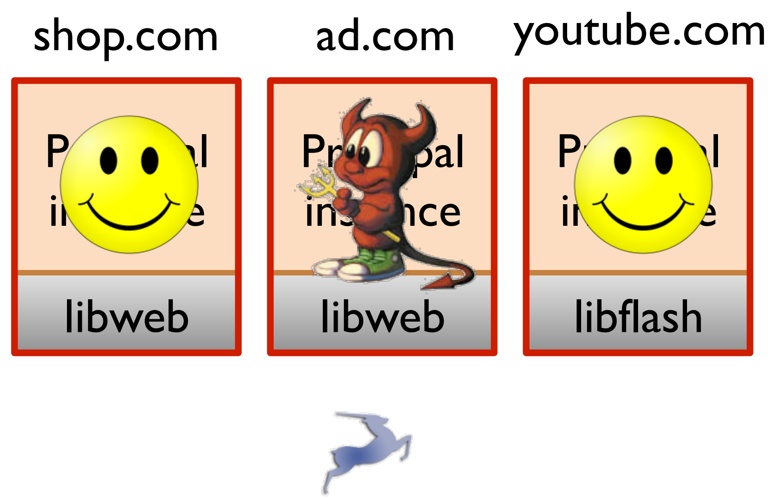
- Principal instances:
  - perform all content processing
  - access resources through system calls to browser kernel
  - reside in sandboxed processes

Architecture



- Principal instances:
  - perform all content processing
  - access resources through system calls to browser kernel
  - reside in sandboxed processes

Gazelle's security and robustness benefits



IE 8 & Google Chrome:

- Security goal: protect host machine
- Multiple processes are for reliable browsing sessions
- Security decisions made in rendering process

Rendering process for shop.com

Architectural implications

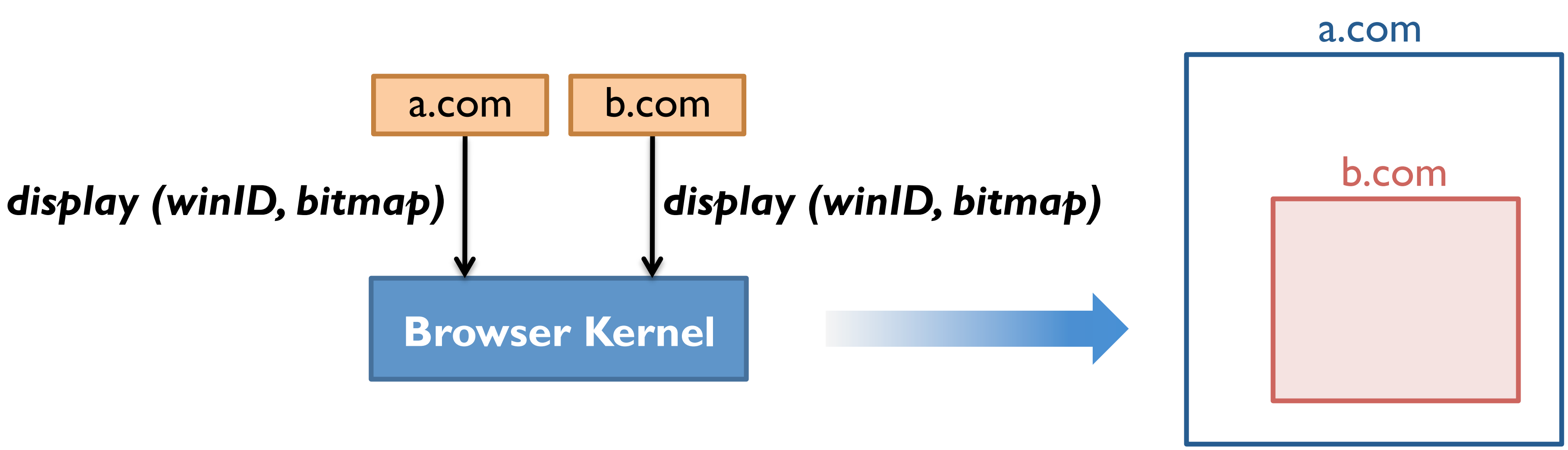
- Gazelle naturally provides principal-based isolation for:
  - *all* resources
    - network, display, memory, persistent state, etc.
  - *all* types of web content
    - HTML, JavaScript, images, *plug-in content*, etc.
- This differs from today's browser policies
  - often inconsistent across resources
  - e.g., cookies, scripts (document.domain exception)
- Achieving backward compatibility is a policy issue
  - can achieve through cross-principal communication
  - this work: focus on architectural issues
  - future work: balance backward compatibility and security

Embedding cross-principal content

- Powerful paradigm in modern web
- Key deviation from the desktop model
- Implications for browser's resource allocation:
  - display (next)
  - other resources: CPU, memory, network (not in this talk)

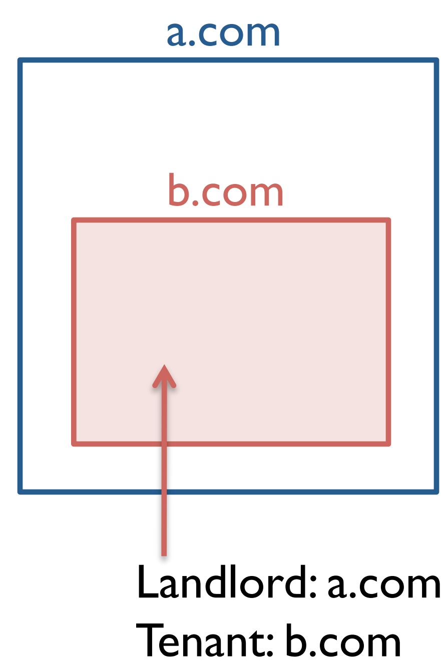
Display in Gazelle

- Browser kernel manages the display
  - browser kernel doesn't know content semantics
- Principal instances render the content
- Browser kernel composes the display



Dual ownership of a window

- ***Window***: unit of display delegation
- A window has *two* owners:
  - *landlord*: creator principal
  - *tenant*: resident principal
  - landlord can delegate screen area to tenant
    - *delegate()* system call
- Window's resources:
  - position, dimensions, content (pixels), URL location



Display Access Control

	Position	Dimensions	Pixels	URL location
Landlord	RW	RW		W
Tenant		R	RW	RW

Tenant cannot tamper owner's display

Tenant's display content

No navigation history leakage

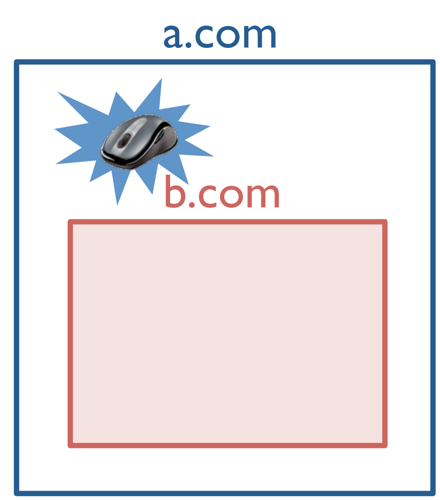
Unlike existing browsers, display ownership and access control is managed exclusively by the browser kernel

Protecting events

- Browser kernel must dispatch user events to the right principal instance

- Challenge: cross-principal content overlaying
  - frames can be transparent
  - images under text
  - layers in CSS

- Layout policy has security and backward compatibility implications
  - see paper



Outline

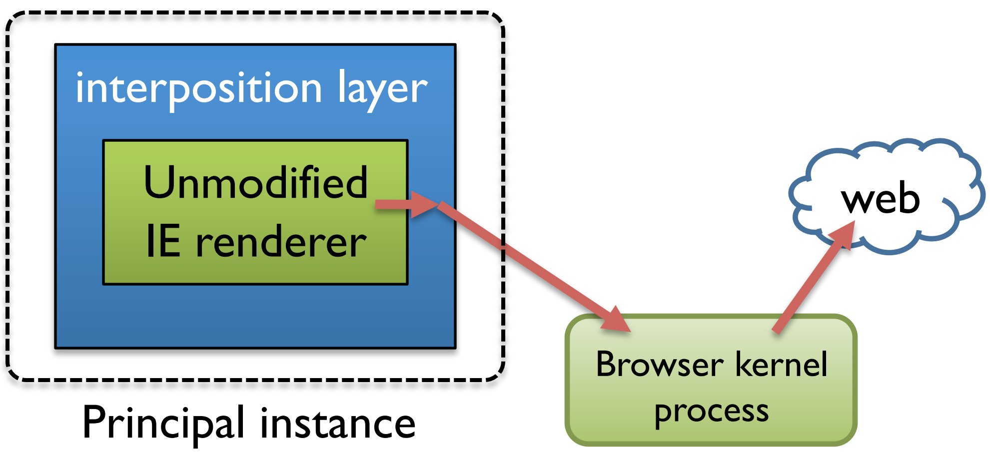
- Motivation
- Gazelle's design
- Implementation
- Evaluation

Implementation

- Browser kernel implemented in C# (5K lines)
- Principal instance is based on Internet Explorer renderer
  - we reuse IE rendering engine (Trident)
    - "free" HTML parsing/rendering, JavaScript engine, DOM
  - Trident instrumented to redirect resource access to BK
  - sandboxed process simulated through interposition
  - no plug-ins yet

Implementation

- Browser kernel implemented in C# (5K lines)
- Principal instance is based on Internet Explorer renderer
  - we reuse IE rendering engine (Trident)
    - "free" HTML parsing/rendering, JavaScript engine, DOM
  - Trident instrumented to redirect resource access to BK
  - sandboxed process simulated through interposition
  - no plug-ins yet



Evaluation – browsing performance

- On par with IE7 and Chrome when browsing within an origin

Examples of web page load times	Gazelle	IE7	Chrome
Navigate from google.com to google.com/ads	1.0s	1.1s	1.0s

Evaluation – browsing performance

- On par with IE7 and Chrome when browsing within an origin
- More overhead for cross-origin navigation, rendering embedded cross-origin content
  - main source: IE instrumentation
    - 1.4s for nytimes (55% of overhead over IE7)
    - can be eliminated in a production implementation

Examples of web page load times	Gazelle	IE7	Chrome
Navigate from google.com to google.com/ads	1.0s	1.1s	1.0s
Navigate from google.com/ads to nytimes.com	5.8s	3.2s	3.5s

(nytimes.com contains a cross-origin iframe)

Evaluation – browsing performance

- Many other optimizations can bring performance on par
  - overlapping fetching and rendering
  - pre-initializing principal instance processes
  - named pipes
  - bitmap transfers

Examples of web page load times	Gazelle	IE7	Chrome
Navigate from google.com to google.com/ads	1.0s	1.1s	1.0s
Navigate from google.com/ads to nytimes.com	5.8s	3.2s	3.5s

(nytimes.com contains a cross-origin iframe)

Related work

- Security in other browsers:
  - IE8/Chrome: different security goals
    - origin protection logic in rendering process, not in browser kernel
  - OP: additional modularization without security benefits
    - some OS logic (e.g., display management) still in principal space
  - Tahoma: VM isolation, web sites opt in to take advantage
  - SubOS: principal definition differs from today's web
- None of these handle embedded web principals

Ongoing research

- Compatibility vs. security cost analysis
  - large-scale study over real web sites
- Sandboxing for principal instances
  - system call interposition (Xax)
  - binary rewriting (Native Client)
  - adding native OS process sandboxing support
- Porting plug-ins into the Gazelle architecture
- Secure device access
- Resource scheduling

Concluding remarks

- Gazelle:
  - protect online valuables
  - first multi-principal OS-based browser design
    - OS functions shifted from renderer to privileged browser kernel
    - browser kernel *exclusively* manages principals, resources, and security
  - architecture does not prevent backward compatibility
  - cross-origin mashups present intricacies in managing display and other resources
- Practical to turn an existing browser into Gazelle

Questions?