The Security Architecture of the Chromium Browser

Charlie Reis

Adam Barth, Collin Jackson, The Google Chrome Team

Stanford Security Seminar, December 2, 2008

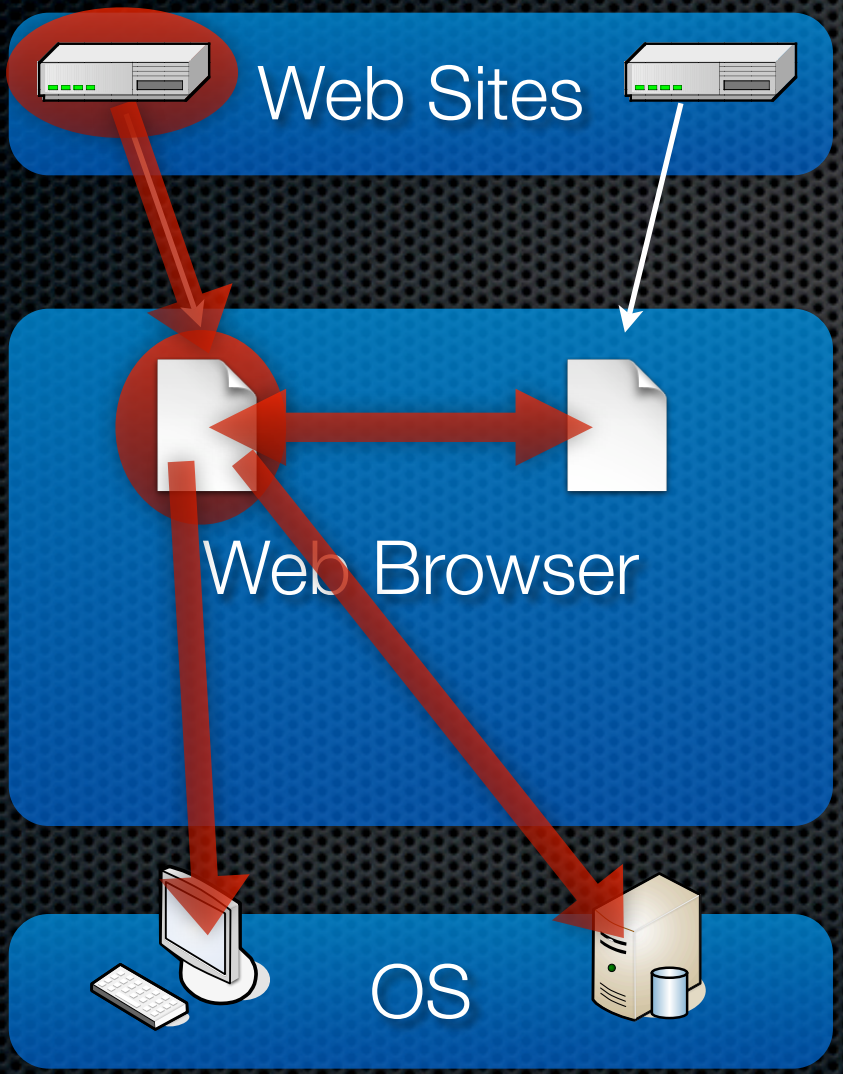
Web is Evolving

- More complex, active content
- More attack surface: vulnerabilities at many levels

Pages

Programs

Attacks at Many Levels



Phishing,
Web Site Vulns

Web Site Isolation

Malware, File Theft,
Keylogging

Browser Exploits

Browser Exploits

- How much damage can they cause?
- Can the browser's architecture reduce it?

Impact of a Page Visit

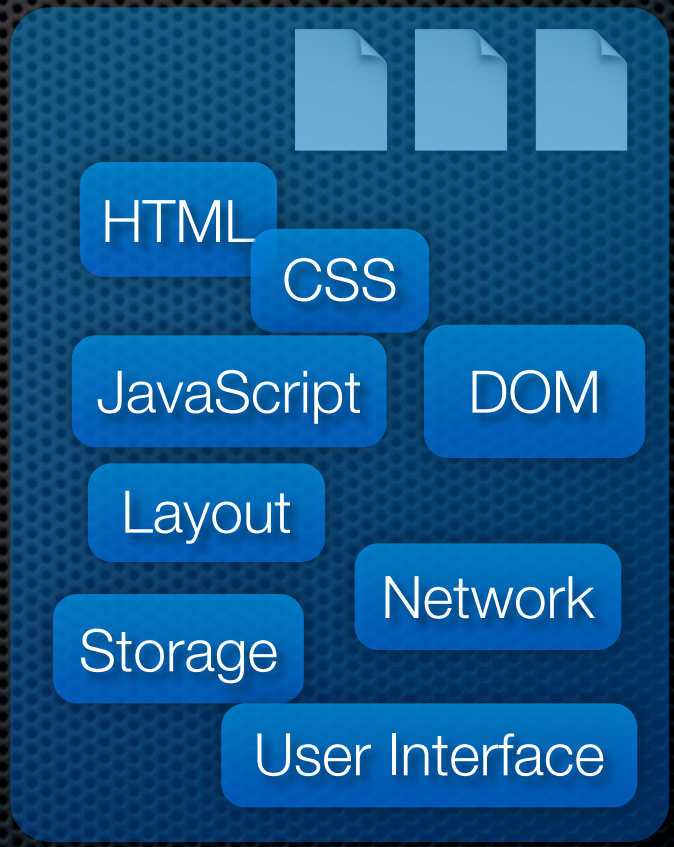
- **Normally:**
  - Leave cookies, cached objects
  - Communicate with servers
  - Downloads, uploads, use devices
- **With an exploit:**
  - Install malware, steal files, log keystrokes
  - Access user's web accounts

Exploits aren’t going away

- Browsers are complex, evolving
- Unsafe languages
  - Massive barrier to entry, so unlikely to change
- Tools can help, but still let bugs through
- Money in malware

Limit the Damage

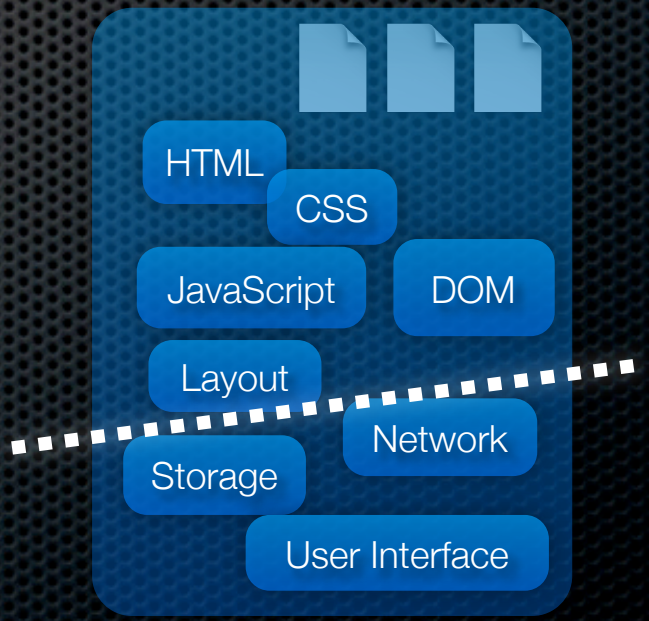
- **Most browsers are monolithic**
  - Rely on logic, not architecture
  - Often have full privileges of user
- **Architecture could help**
  - OSes isolate users, VMs isolate untrusted code, etc



One Protection Domain

Modularize the Browser

- **Don’t run all parts of browser with full privileges**
  - Some parts more likely to be hacked than others
- **Use privilege separation**
  - Limit impact of many exploits



Outline

Motivation

Overview

Chromium's Architecture

Security Evaluation

Going Further

Chromium's Approach

- **Divide browser into modules:**
  - Browser kernel (runs as "the User")
  - Rendering engine (runs as "the Web")
- **Focus on:**
  - Compatibility with existing content
  - Treating rendering engine as a black box

Threat Model

- Assume attacker will exploit your browser
- **In scope: protect the user principal**
  - Malware
  - Keylogging
  - File Theft

Threat Model

- **Out of scope: protect user's web accounts**
  - Phishing
  - Web site vulnerabilities (XSS, etc)
  - Violating Same Origin Policy

Related Browsers

- **Monolithic (Popular)**
  - Firefox 3, Safari 3: *full user privileges*
  - IE 7: *protected mode (read, but no write)*
- **Modular (Proposed)**
  - SubOS, DarpaBrowser, Tahoma, OP: *break compat*
  - IE 8: *multi-process, still allows file theft*

Outline

Motivation

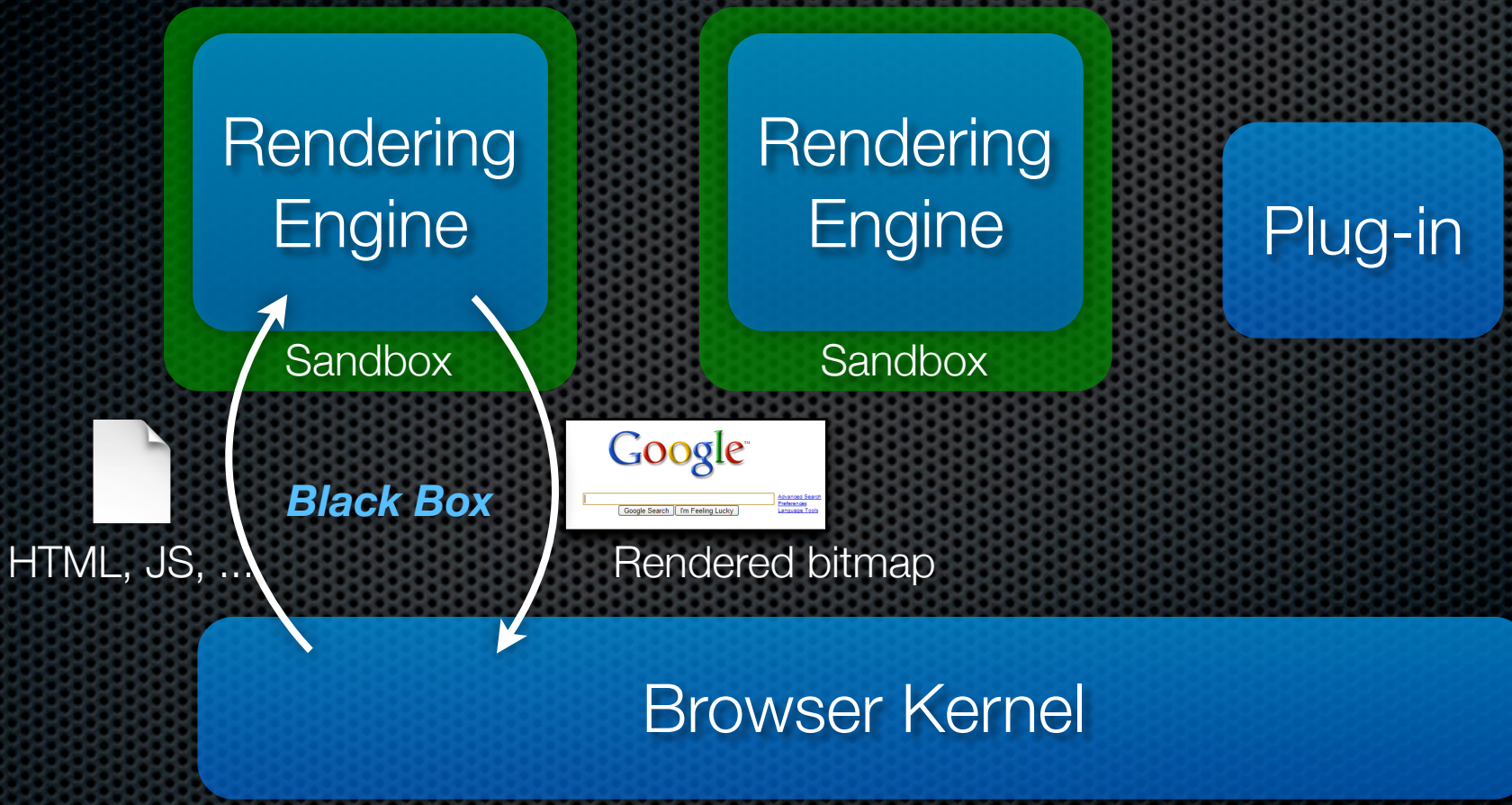
Overview

Chromium's Architecture

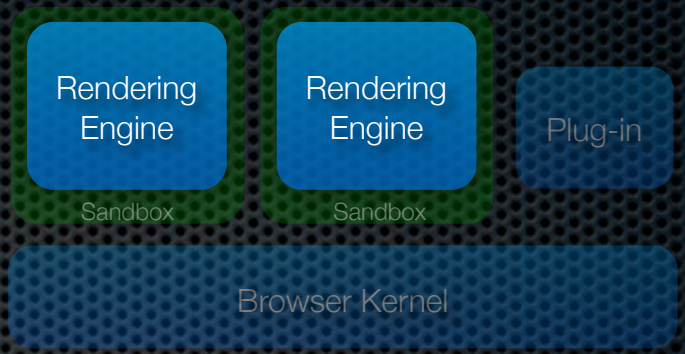
Security Evaluation

Going Further

Chromium's Architecture

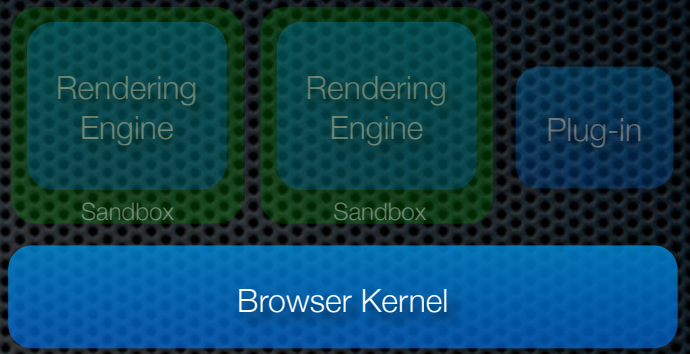


Rendering Engine



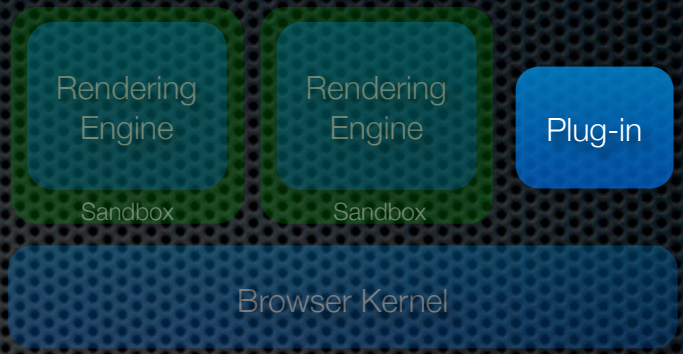
- **Render HTTP responses into bitmaps**
  - Parse HTML, CSS, SVG, XML, etc
  - Manage DOM and layout
  - Interpret scripts, decode images
- **Most complex, most attack surface**
  - Run inside sandbox to reduce privileges

Browser Kernel

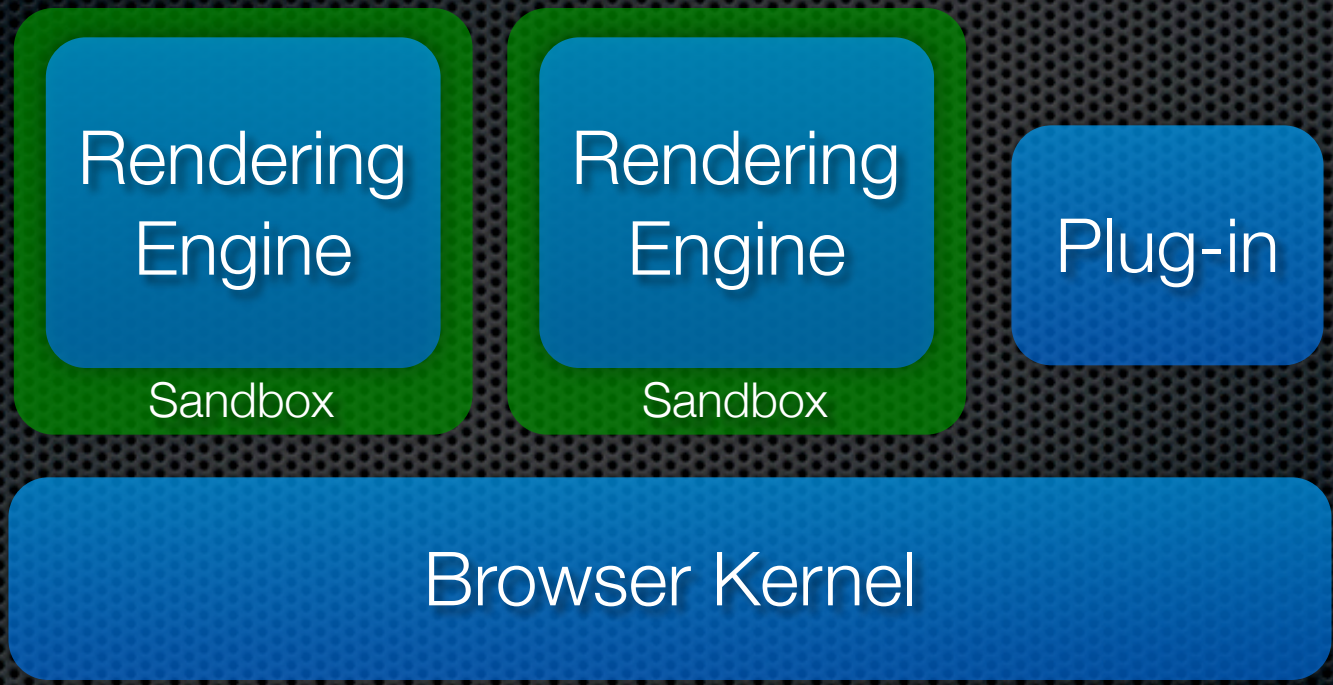


- **Interact with user and operating system**
  - Window management, location bar
  - Storage of cookies, history, cache, downloads
  - Network stack
- **Enforces policies on rendering engines**

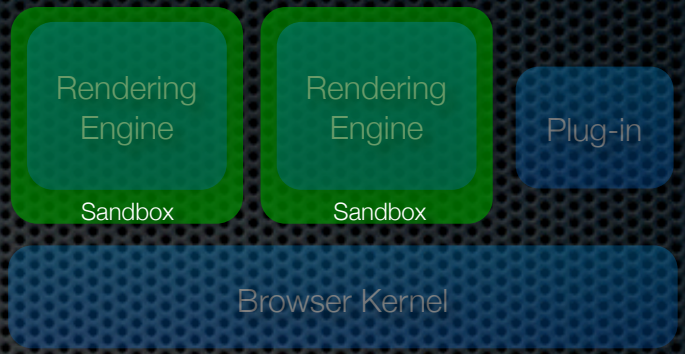
Plug-ins



- **Pose a Dilemma:**
  - Widely used, but not under browser's control
  - Don't want in browser kernel (reliability)
  - Can't easily be sandboxed (compatibility)
- **Put in own process, one per plug-in type**
  - Doesn't address security
  - *Could plug-ins move to a new model?*

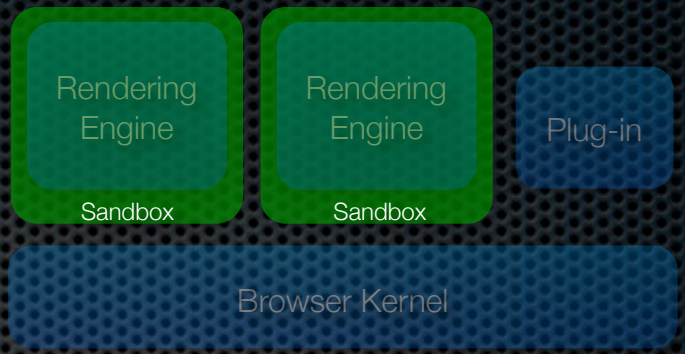

Rendering Engine
Sandbox
Rendering Engine
Sandbox
Plug-in
Browser Kernel

Sandbox



- **Goal:** can't affect world, except via exposed API
  - Block access to all objects, resources
  - Not trying to block system calls
- **Approach:**
  - Start process, establish IPC channel
  - Drop all access privileges
  - *Don't require admin rights*

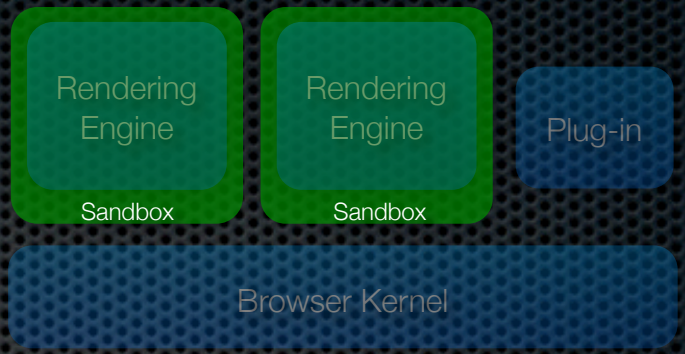
Implementation
(on Windows)



1. **Restricted security token**
2. **Job object**
3. **Separate Desktop object**

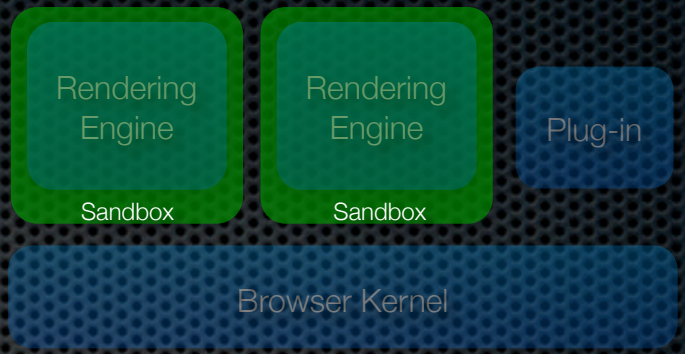
See also: David LeBlanc's blog

Restricted Token



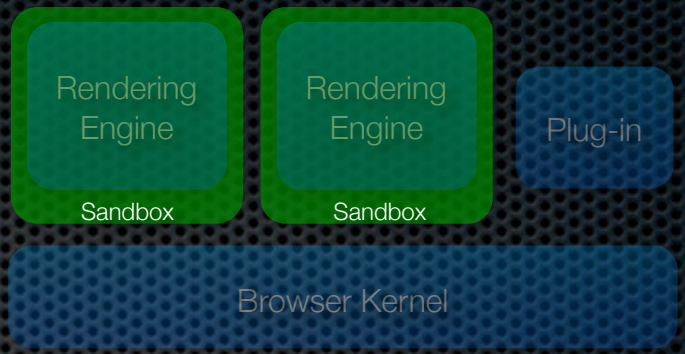
- **Prevents access to (almost) all resources**
- Derived from user's security token
  - Works with existing auditing systems
- Vista: also uses a "low integrity level" label

Job Object



- **Restricts actions other than resource access**
  - Can’t create processes or desktops
  - Can’t change system settings or log off
  - Can’t access clipboard, etc.

Separate Desktop



- **Receives no input events from user**
- **Prevent messages to more privileged windows**
  - Avoids “shatter attacks” that inject code
- **One desktop for all sandboxed renderers**
  - Safe: renderers have no windows

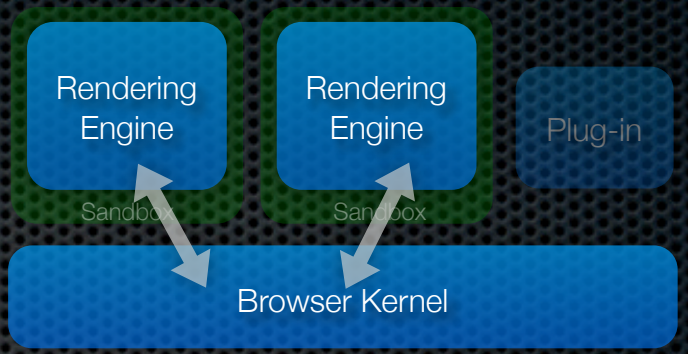
Sandbox Limitations

- Can still access FAT32 drives
- Can still access some misconfigured objects *(if they have null DACLs)*
- Theoretical access to TCP/IP on Windows XP

Sandboxed Renderers

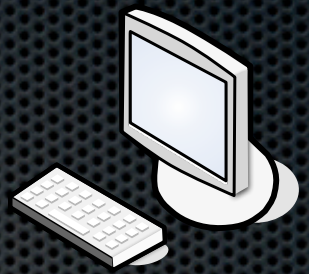
- Sandbox itself is general purpose
- **Straightforward to sandbox WebKit**
  - Platform-specific glue layer:
    talk to browser kernel

Browser Kernel API



- How renderer influences outside world
- Exposes UI, storage, network
- Chance to enforce policies on renderer behavior

Browser Kernel API

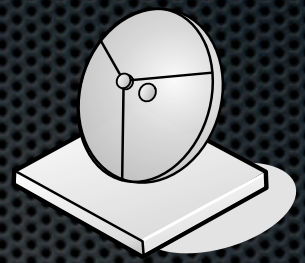


- **User Interaction**
  - Display rendered bitmaps
  - Forward input events

Browser Kernel API

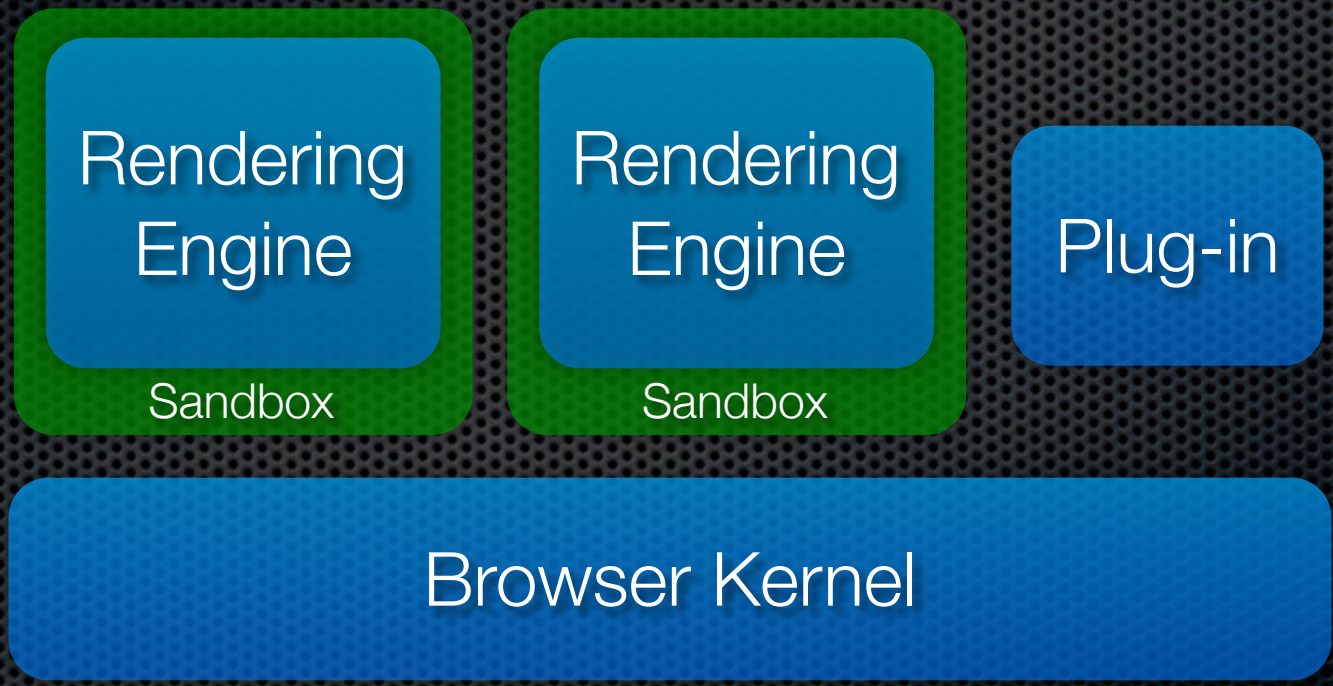
- **Storage**
  - Manage cookies, passwords, etc
  - Authorize uploads
  - Restrict downloads

Browser Kernel API



- **Network**
  - HTTP Requests and Responses
  - Restrict certain schemes (e.g., `file://`)

Summary


Rendering Engine
Sandbox
Rendering Engine
Sandbox
Plug-in
Browser Kernel

Outline

Motivation

Overview

Chromium's Architecture

Security Evaluation

Going Further

Challenging to Evaluate

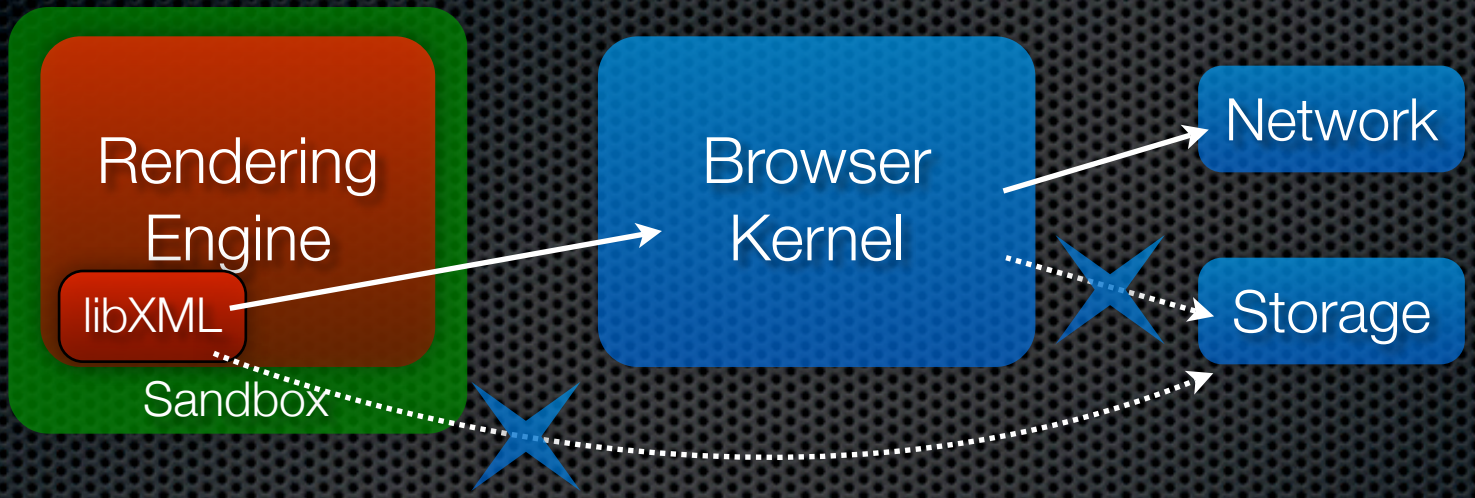
- Hard to reason about all possible attacks
- Instead:
  - Look at a case study of how it has helped
  - Generalize from past vulns. in other browsers

Case Study: XXE

XXE Vulnerability

- **XML External Entities**
  - Define your own entities, like `&copy;` for ©
  - Fetch from a file or URL
- **Vuln in libXML**
  - Attackers could fetch from filesystem or other origins

Impact in Chromium



- **libXML lives in rendering engine**
  - Cross-origin requests were possible
  - Browser kernel blocked access to disk

Vulnerability Analysis

Vulnerability Analysis

- Chromium is new, so not many vulnerabilities to study
  - Look at other popular browsers
- **Questions:**
  - Which modules tend to be more vulnerable?
  - Where are the biggest threats?
  - Is Chromium's architecture focusing on right parts?

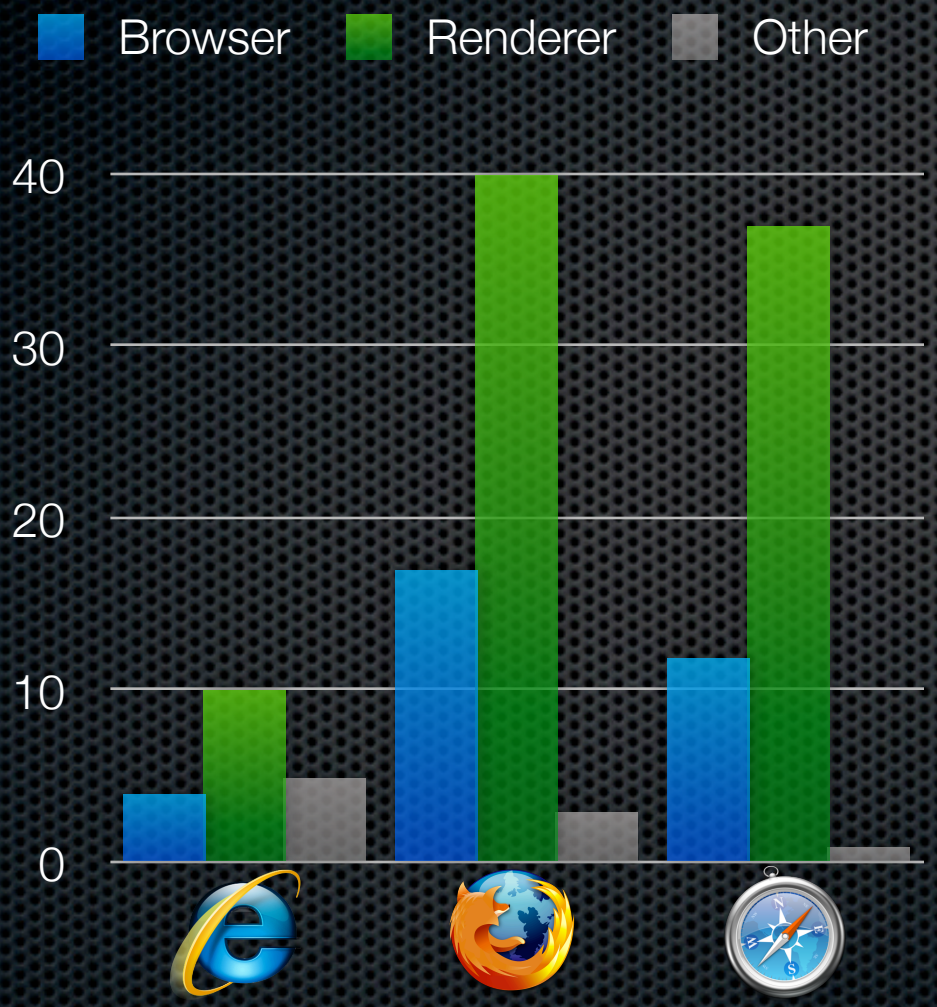
Past Vulnerabilities

- Studied IE, Firefox, and Safari vulns from past year *(can't compare directly; different methodologies)*

Internet Explorer	19
Firefox	60
Safari	50

- Categorize vulns by Chromium module

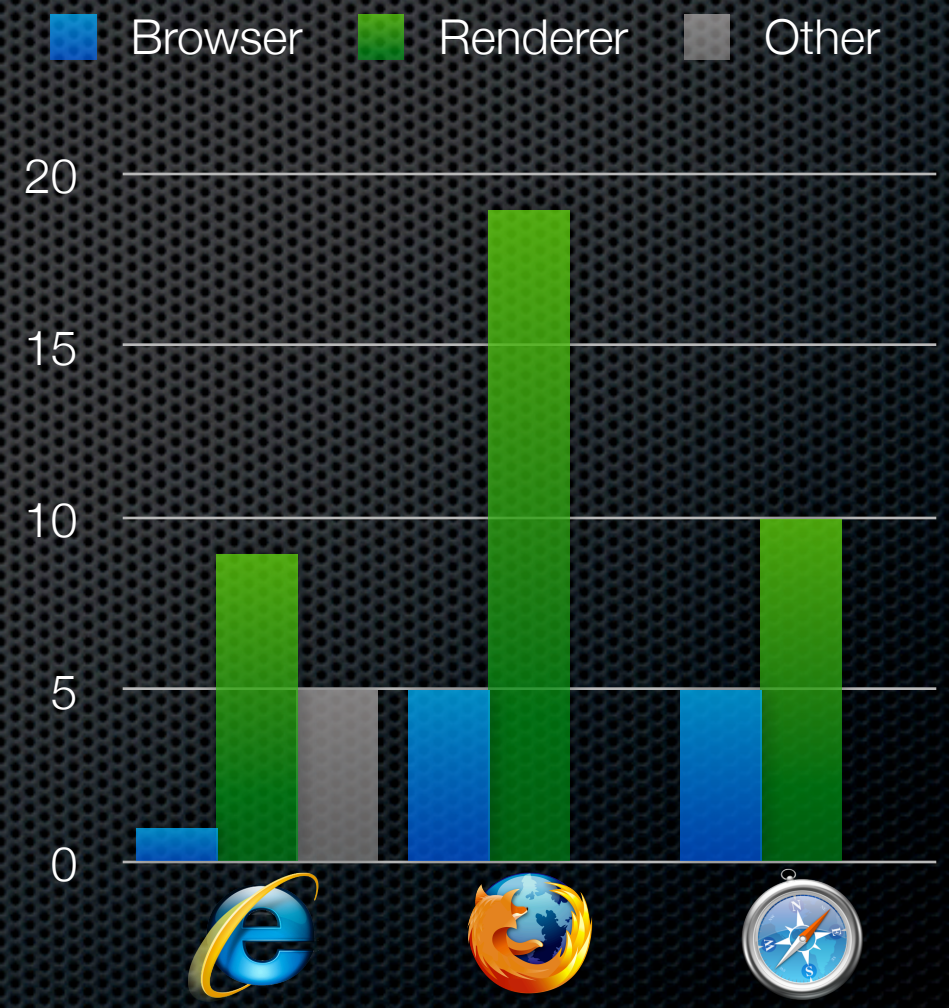
Which modules have vulns.?



- Renderers *twice* as vuln as browser kernels
  - Complex
  - Worthy of attention

Where are the worst vulns.?

- **Arbitrary code execution**
- Renderers have twice as many as browser kernels
- Sandbox would mitigate
  - Block malware, keyloggers, file theft



Remaining Vulns.?

- 11 ACE vulnerabilities in browser kernels
  - 8 of these: insufficient validation of OS calls
  - Sandbox wouldn’t help
- *Getting good mileage from sandbox*

Summary

- Rendering engines vulnerability prone
- Sandbox helps with most of the worst vulnerabilities

Outline

Motivation

Overview

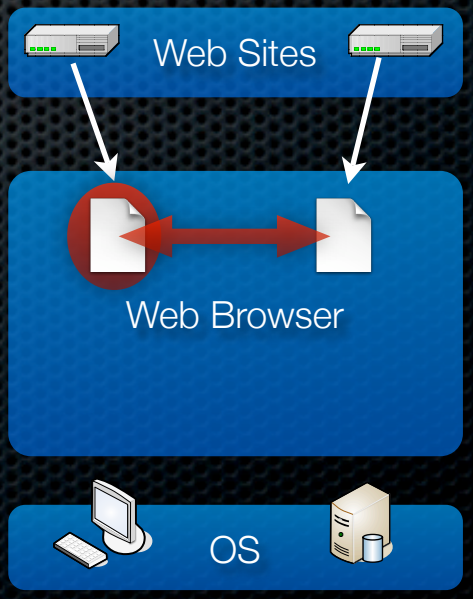
Chromium's Architecture

Security Evaluation

Going Further

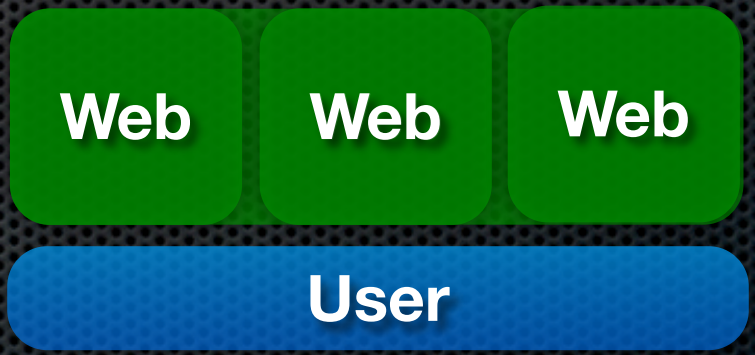
Revisit Other Threats

- **Phishing:**
  - User perception issue, use blacklists
- **Web site vulns:**
  - Some research, rely on sites
- **Web site isolation:**
  - Room for improvement



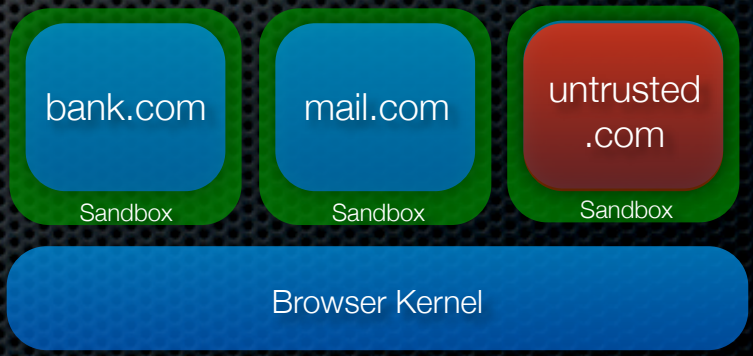
Site Isolation

- Want to protect web site accounts: banks, mail, etc.
  - **Web principals:** web site + user's credentials
- Can we enforce isolation despite renderer exploits?



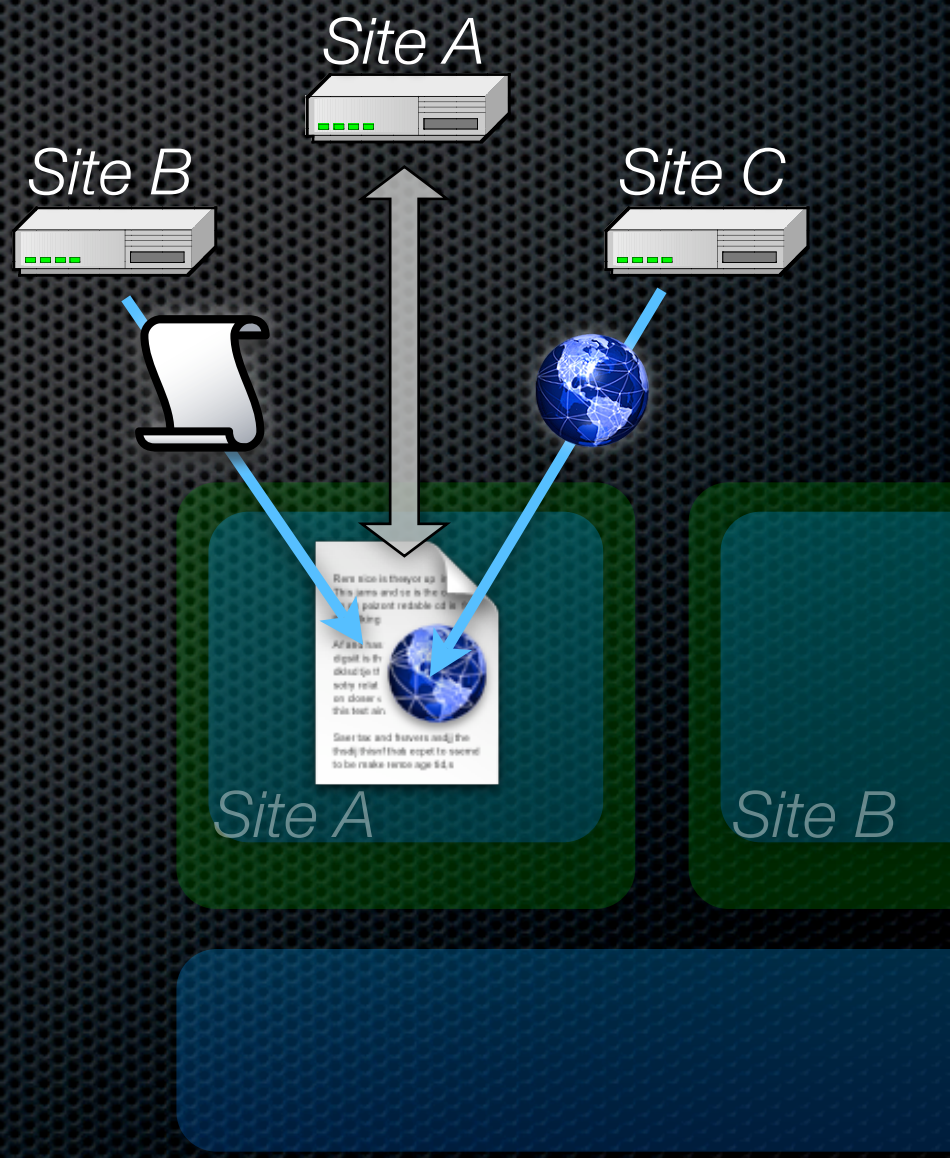
Rendering Engine Isolation

- **Already have multiple rendering engines**
  - Reliability, performance
- **Separate pages based on site?**
  - Let sandbox isolate them
  - Chromium partly there



Freedom is a Challenge

- **Pages are free to embed objects from any site**
  - Images, scripts, frames...
  - Carry user's credentials
  - *Sensitive info in renderer*
- **Black box:**
  don't split out sub-objects
- **Compatibility:**
  don't block credentials



Future Work

- For now, rely on rendering engine's logic
- Look at ways to isolate web principals, while preserving compatibility

Conclusion

- **Browser's architecture can mitigate many exploits**
  - Limit privileges of rendering engines
  - Help prevent malware, keyloggers, file theft
- Opportunities for protecting web principals