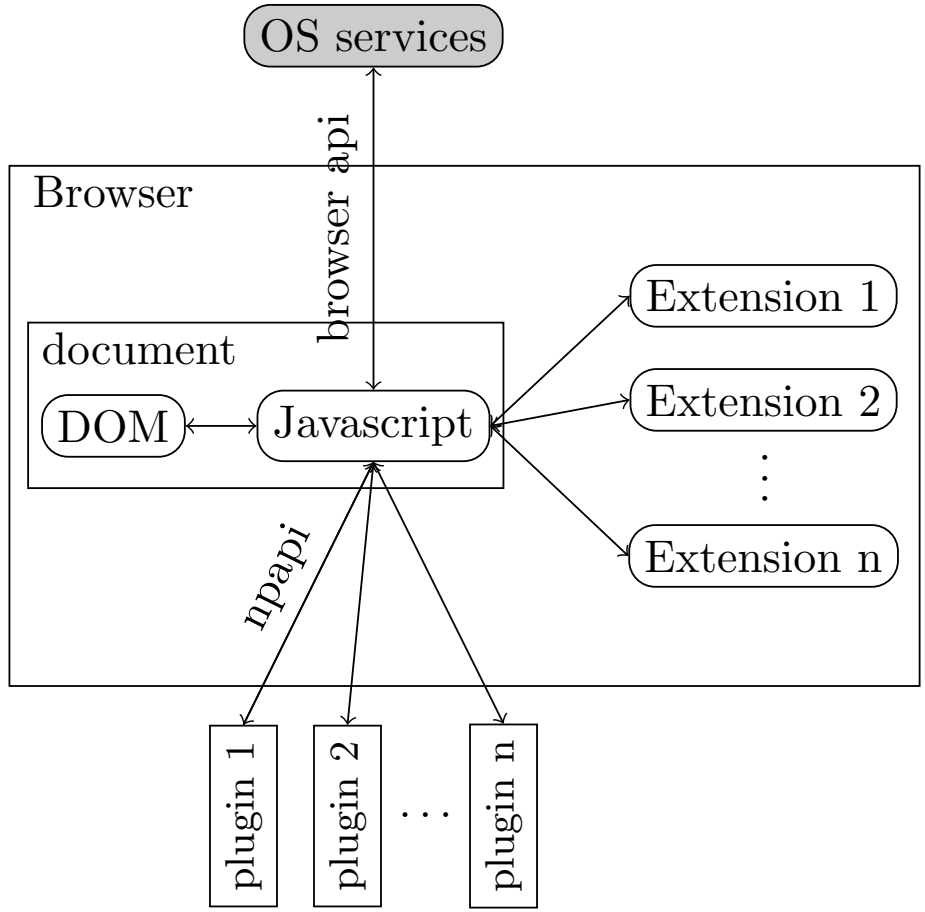
Security of web browsers I

November 7, 2019

Motivation

How frequently do you run new and untrusted code?

Modern browsers are complex



Doom in the browser
Technologies in cnn

Browser vulnerability

Since the modern browser is very complex relying on many 3rd party libraries, there is a high chance there will be a bug.

Example of buffer overflow

CVE-2016-1681 Default pdf reader in Google Chrome web browser had an exploitable heap buffer overflow vulnerability. By simply viewing a PDF document that includes an embedded jpeg2000 image, it is possible to execute arbitrary code.

Example of use-after-free

CVE-2016-2821: Use-after-free vulnerability in the mozilla::dom::Element class in Mozilla Firefox before 47.0 and Firefox ESR 45.x before 45.2, when contenteditable mode is enabled, allows remote attackers to execute arbitrary code or cause a denial of service (heap memory corruption) by triggering deletion of DOM elements that were created in the editor.

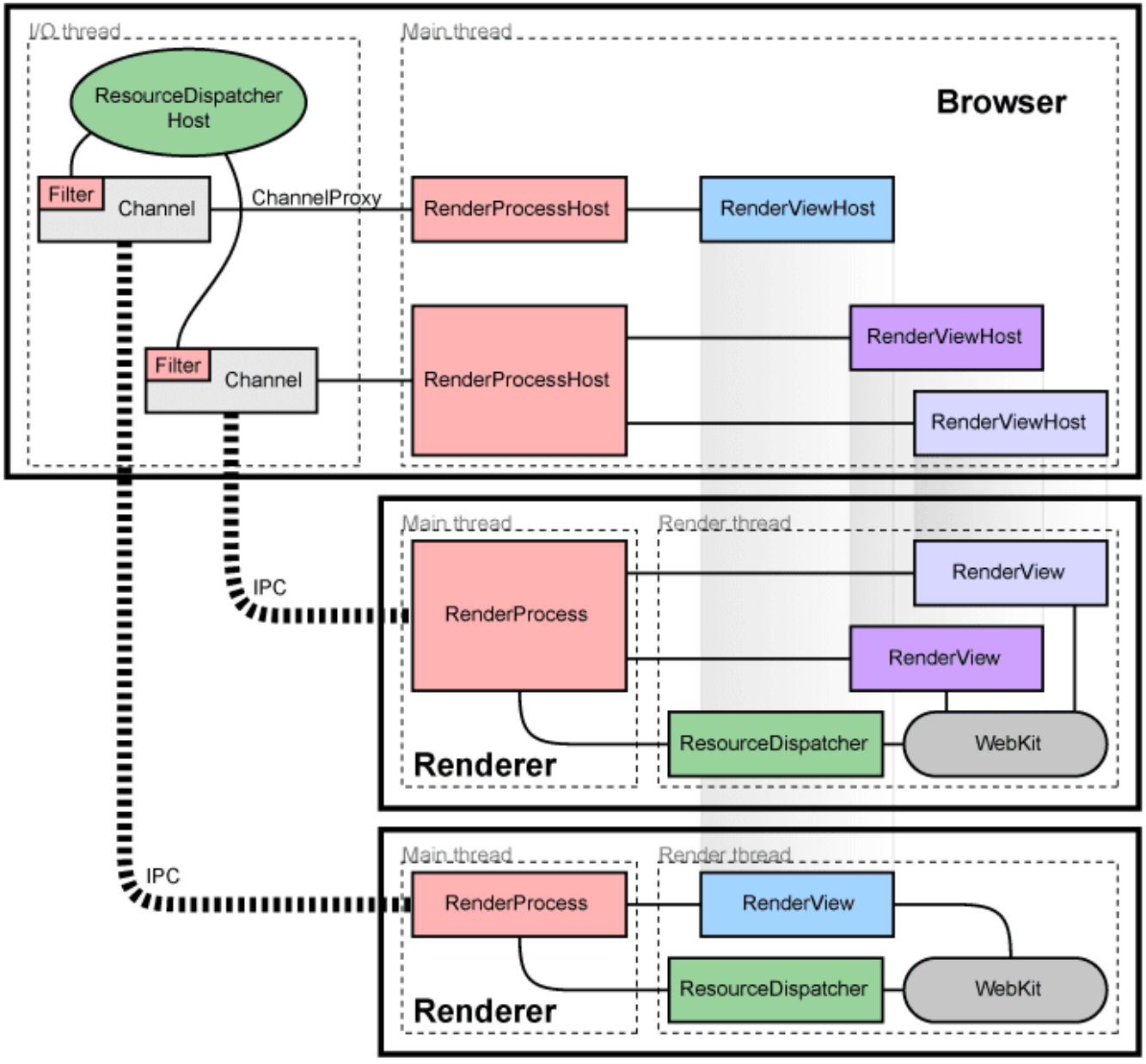
Example of integrity attack

CVE-2016-2819: Mozilla windows updater does not lock files for writing and can be overwritten by other process with their own content while updater is running. This vulnerability could be used for privilege escalation if these overwritten files were later invoked by other Windows components that had higher privileges.

Plan

Sandboxing browsers

Security architecture of chromium



Division of tasks

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

Sandboxing in chrome

Chrome runs each panel as a separate process with restricted api

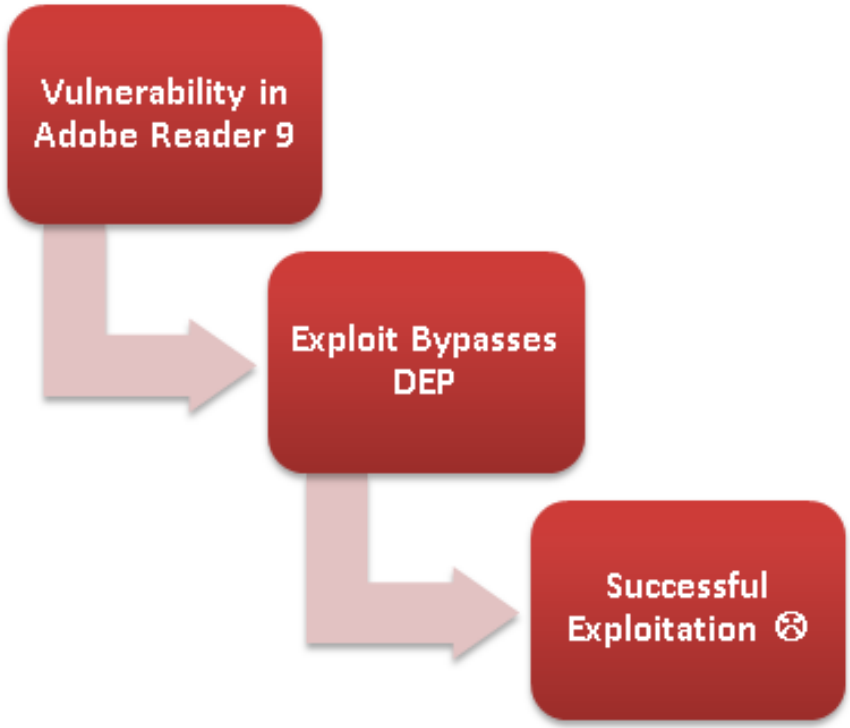
Linux

- empty root
- process namespace
- network namespace
- syscall whitelist
- seccomp-bfl

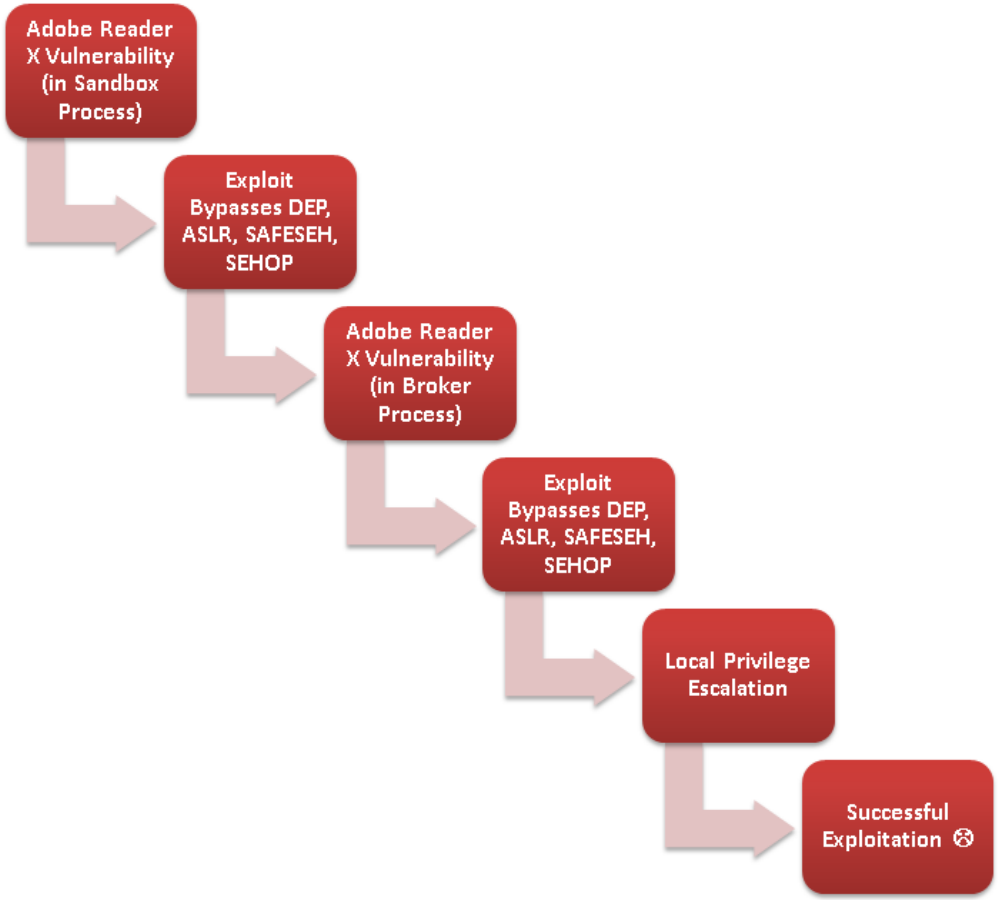
Windows

- restricted access tokens
- job object limitations
- window station and desktop Isolation
- mandatory integrity control
- exports API for sandboxed applications

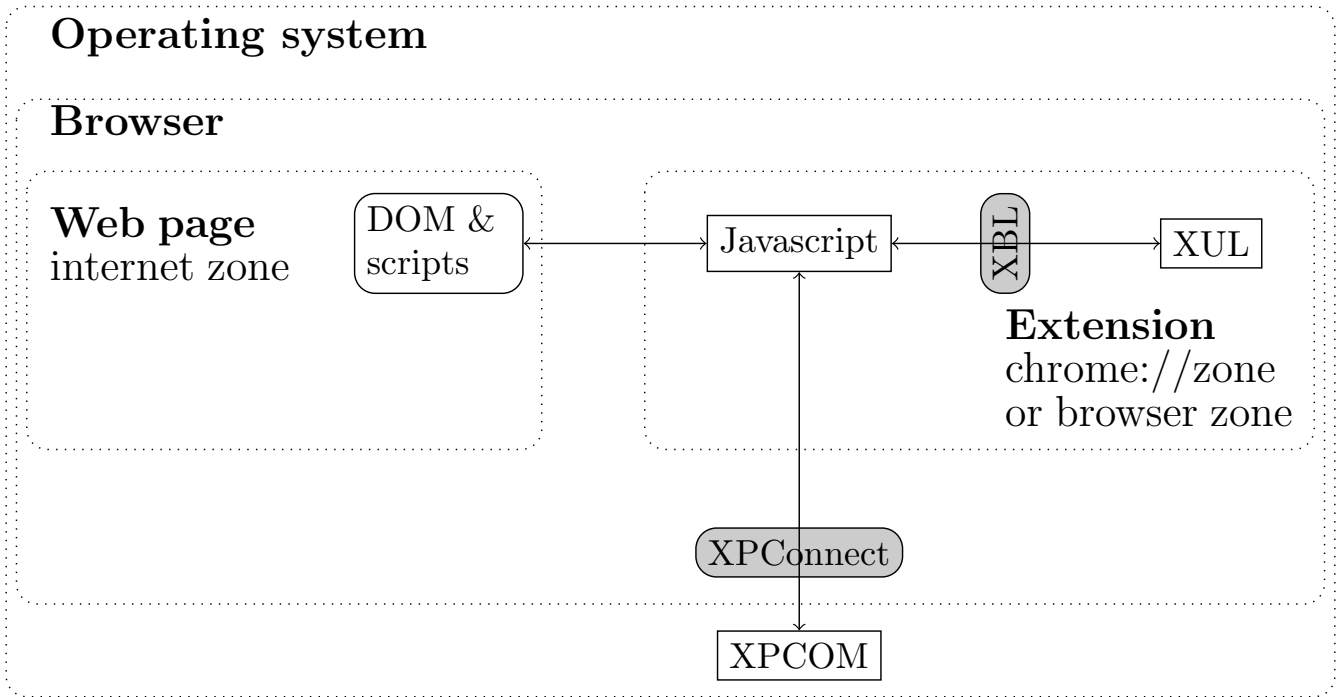
Effect of sandboxing (in Acrobat) Win XP



Effect of sandboxing (in Acrobat) Win 7



Extensions in Firefox 2003-2015

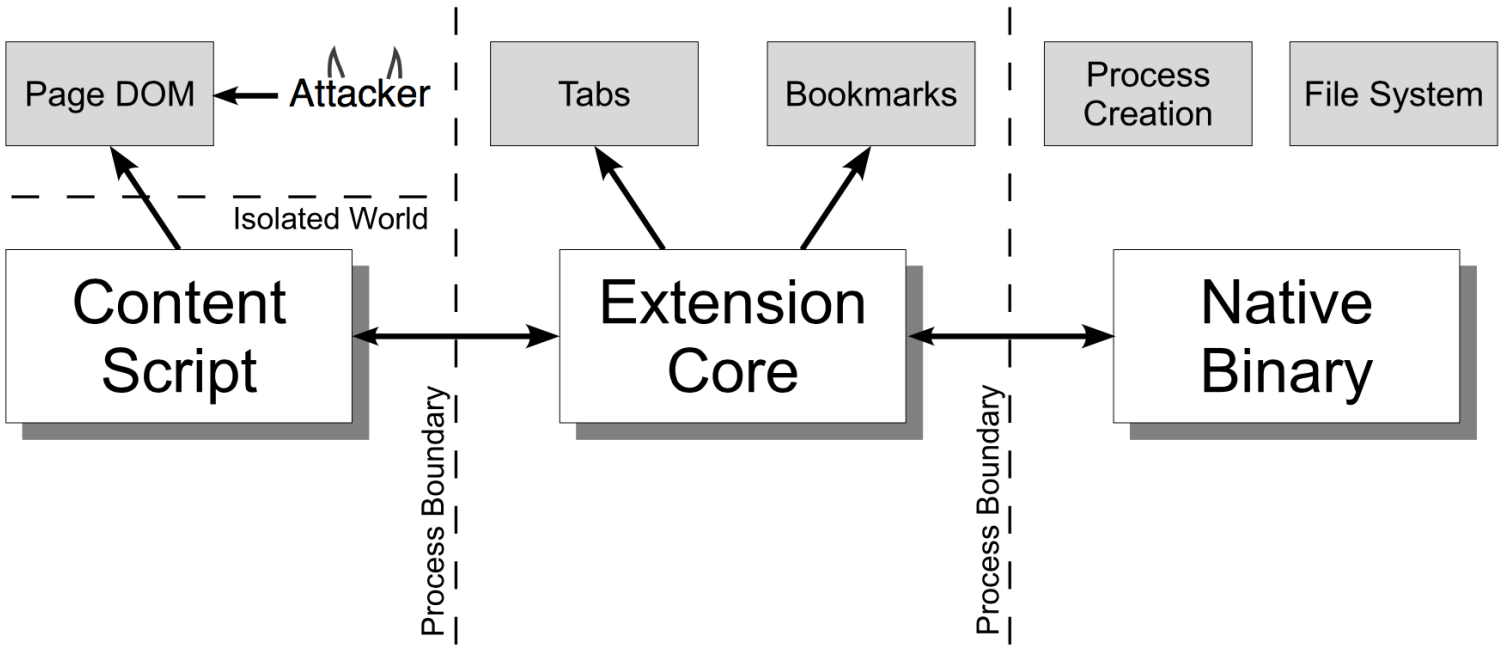


chrome://browser/content/browser.xul

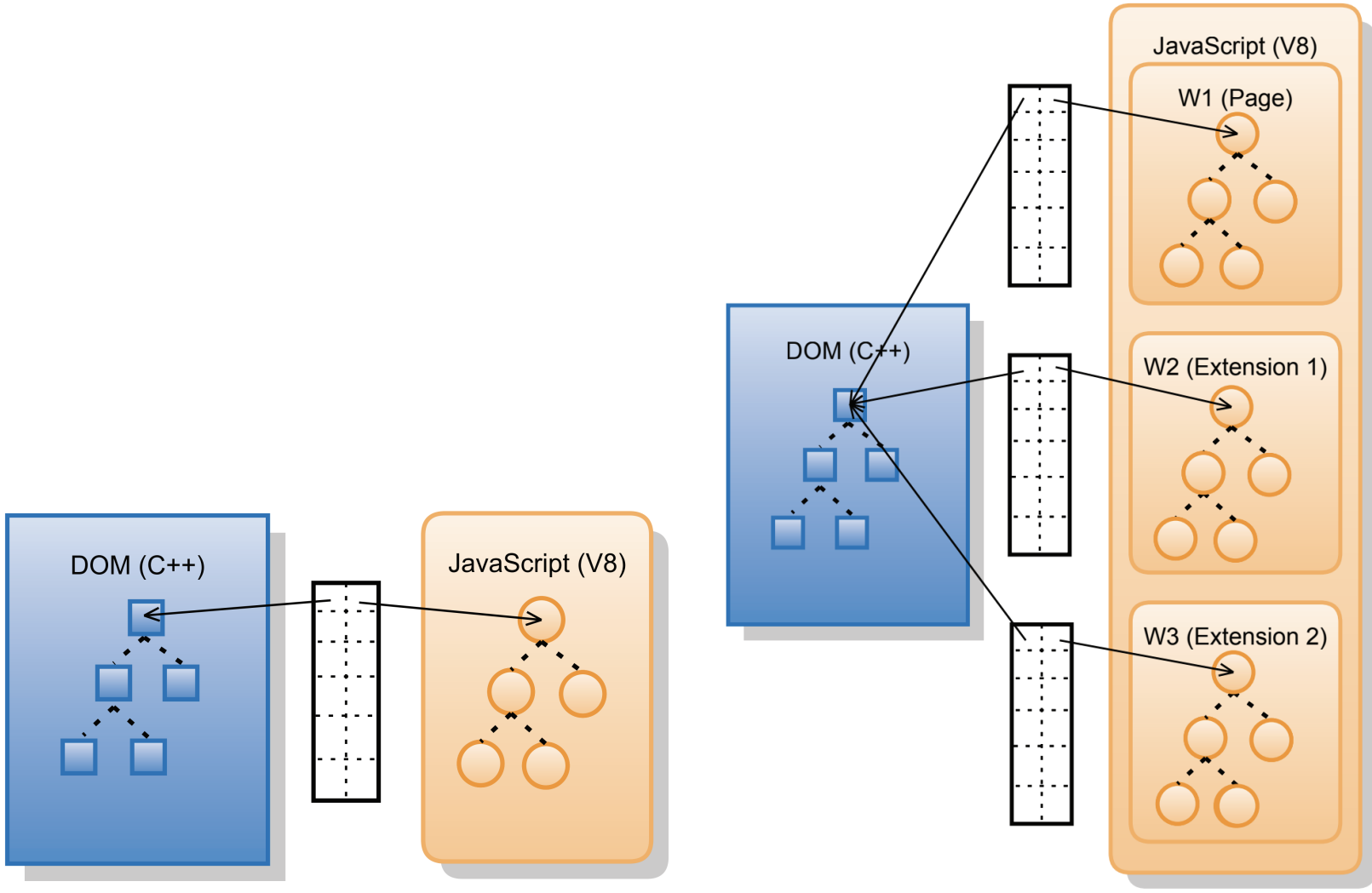
Example of manifest.json

```
{
"name": "Google Mail Checker",
"description": "Displays the number of unread
   messages...",
"version": "1.2",
"background_page": "background.html",
"permissions": [
  "tabs",
  "http://*.google.com/",
  "https://*.google.com/"
],
"browser_action": {
    "default_title": ""
},
"icons": {
  "128": "icon_128.png"
} }
```

Chrome extension model



Isolating worlds



Security evaluation of chrome's extensions

From study[1]

- *Isolated worlds* is highly effective.
- *Separation* would protect 62% percent in the case the above fails.
- Permissions significantly reduce severity of half of the core extensions vulnerabilities.

[1] An Evaluation of the Google Chrome Extension Security Architecture, Nicholas Carlini, Adrienne Porter Felt, and David Wagner, 2012

Threats to extensions

- Data as HTML
- Click injection
- Web Site Metadata Vulnerabilities
- Direct network attack

Google's native client for x86-32

- modified compiler produces code produces code that can be efficiently verified
- address protection relies on x86's segmentation (no overhead on checking)
- write to code segment is prohibited — prevents self-modifying code
- privileged instructions & syscalls are prohibited
- all jumps are 32 bytes aligned
- returns and calls are prohibited
- *trampoline* — mechanism to jump from untrusted to trusted code
- *springboard* — mechanism to jump from trusted to untrusted code

Cryptography in browser

Is cryptography in Javascript (in Browser) possible?

Cryptography in browser

Cryptography in javascript is currently nonsense, since XSS vulnerabilities gives attacker access to all data and all code.

Javascript cryptography considered harmful, Matasano security, 2011