CSE 127: Computer Security

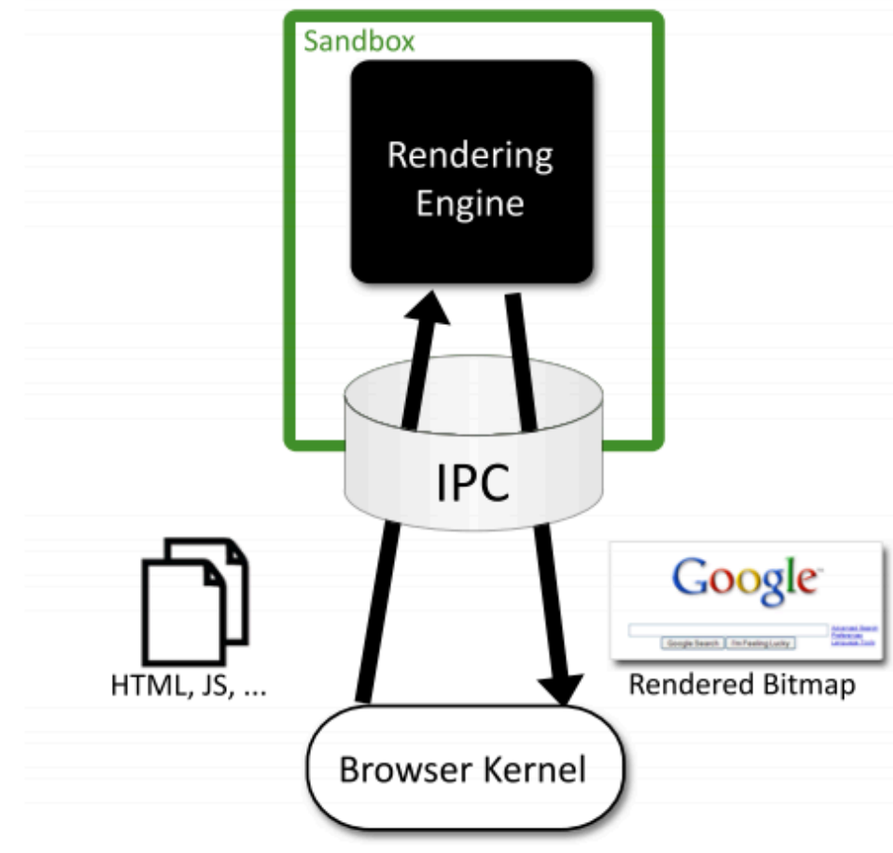
Privilege separation and isolation

Deian Stefan

Slides adopted from John Mitchell, Dan Boneh, and Stefan Savage

Chromium security architecture

- Browser ("kernel")
  - Full privileges (file system, networking)
- Rendering engine
  - Can have multiple processes
  - Sandboxed
- One process per plugin
  - Full privileges of browser



Privilege separation

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

Chrome Security Architecture

Process Level Snapshot

Legend:
- ↔ Chrome IPC
- Minimum Ambient Permissions
- Limited Ambient Permissions
- Elevated Ambient Permissions
- Maximum Ambient Permissions
- Feature not supported on Android

Generic Mitigations:
Process-level sandboxing
DEP+ASLR (per-process on linux & cros)
Stack canaries
Runtime and Library Hardening

Utility Process
Launched for short-lived operations, and will run sandboxed or unsandboxed depending on the specific operation (e.g. printing).

GPU Process
The GPU process runs with the minimum access required for using GPU resources (e.g. low-integrity on Windows).

PPAPI Broker Process
The PPAPI broker is allowed by the user to perform limited privileged actions for the PPAPI process (e.g. update global Flash settings).

Browser Process
The browser process runs at full user privilege and brokers access to most system resources including the profile and any persistent data.

Browser Mitigations:
IPC hardening and CL reviews
Minimal active content (e.g. JS)
Limited protocol parsing

Major Attack Surface:
Web renderer IPC surface
Network protocol parsing
Process state confusion (e.g. navigation)
Google services (e.g. extension syncing)

Renderer Processes

Renderer Mitigations:
Tightest OS sandbox
Scripting runtime
Binding integrity
Memory partitions
Internal origin enforcement*

Major Attack Surface:
Blink
V8 (including RWX JIT)
media (e.g. ffmpeg, libpng)
WebRTC, WebGL, etc.

Extension
Elevated extension and app permissions are listed in manifest file as either optional or required.

WebUI
C++ generated settings and diagnostic pages (effective permissions are hard to quantify).

Elevated Web
Certain web processes implicitly receive limited elevated privileges (e.g. omnibox renderer, Chrome Web Store, file: URLs)

Web
Normal Web content runs at the low privilege, but origins can be granted limited increased permissions by the user.

PPAPI Process
Native code Pepper plugins, including Pepper Flash (which has some elevated APIs).

NaCl Loader Process
Bound by the hosting renderer's origin and an inner SFI sandbox. *(Non SFI code is a work in progress.)*

Sandboxing/isolation techniques

- Layer 1: semantics layer
  - setuid sandbox, prevent access to most resources
- Layer 2: attack surface reduction
  - seccomp-bpf, prevent access to kernel

setuid sandbox (old)

- Creates new network + PID namespace
  - Why?
- Chroot process to empty directory
  - Why?
  - E.g., `chroot /tmp/guest`
    `su guest`
  - `open("/etc/passwd", "r")` translates to...

setuid sandbox (old)

- Creates new network + PID namespace
  - Why?
- Chroot process to empty directory
  - Why?
  - E.g., `chroot /tmp/guest`
    `su guest`
  - `open("/etc/passwd", "r") translates to...`
    `open("/tmp/guest/etc/passwd", "r");`

replacement for setuid sandbox

- Namespaces (Linux v4)
  - mnt
  - pid
  - net
  - ipc
  - user

replacement for setuid sandbox

- Namespaces (Linux v4)
  - mnt
  - pid
  - net
  - ipc
  - user

+ control groups = containers

Layer 2 sandbox: seccomp-bpf

- seccomp - “secure computing mode”
  - no sys calls except exit, sigreturn, read, and write to already open FDs
- seccomp-bpf - syscall filtering
  - allow/deny arbitrary set of system calls
  - filter on syscall arguments
- Why do we want this?

How does seccomp-bpf work?

- Compile BSD packet filters and load them into the kernel
  - Why can't you filter on pointers?
  - Why do it in the kernel?

More general: syscall interposition

- Interpose on system calls
  - Implement agent that does what you want
- Challenges with this approach?
  -
- How do Firefox and Chrome deal with this?
  -

More general: syscall interposition

- Interpose on system calls
  - Implement agent that does what you want
- Challenges with this approach?
  - Keeping state synchronized between kernel and agent
- How do Firefox and Chrome deal with this?
  -

More general: syscall interposition

- Interpose on system calls
  - Implement agent that does what you want
- Challenges with this approach?
  - Keeping state synchronized between kernel and agent
- How do Firefox and Chrome deal with this?
  - Not syscall interposition in pure form, but have trusted parent process broker fs, net, etc. access

- What if we don’t have OS support?
- What if we don’t trust the OS to get this right?

Software-based fault isolation

- You can use SFI to do whole program isolation
  - Google's Native Client did this
- But, what was the original motivation behind SFI?
  -

Software-based fault isolation

- You can use SFI to do whole program isolation
  - Google's Native Client did this
- But, what was the original motivation behind SFI?
  - Sandbox modules/make it easy to extend a program with untrusted code

Software-based fault isolation

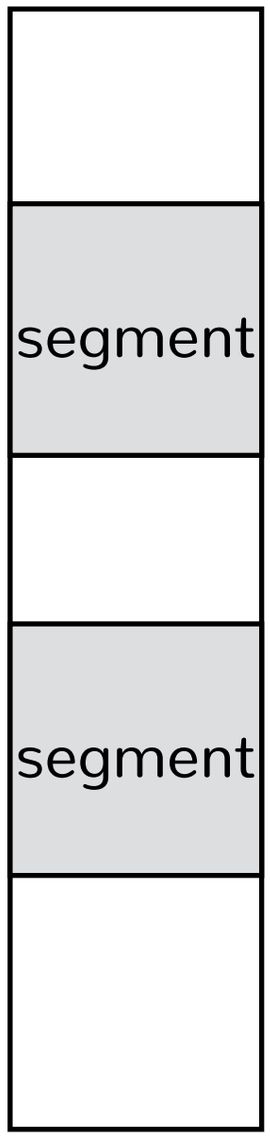
- Can we just do this with OS process isolation?
  - A: yes, B: no
- What's the tradeoff?
  -
  -

Software-based fault isolation

- Can we just do this with OS process isolation?
  - A: yes, B: no
- What's the tradeoff?
  - You often pay context-switch cost
  - Hot-off-the press: with multiple cores you can get SFI and process-based isolation perf to be on par

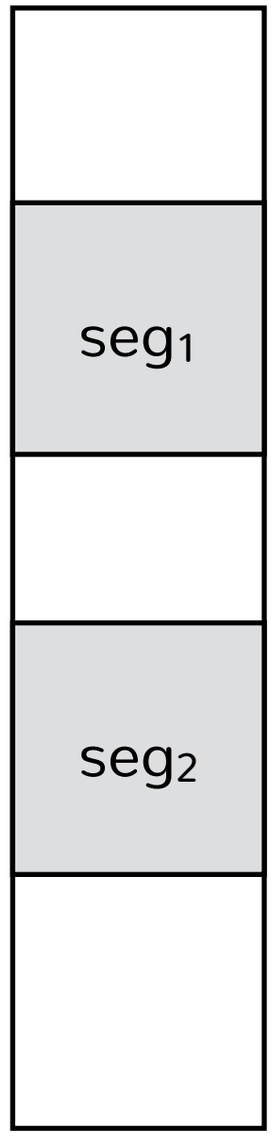
Goals of SFI

- Confidentiality
- Integrity
- Does it provide availability?
  - A: yes, B: no



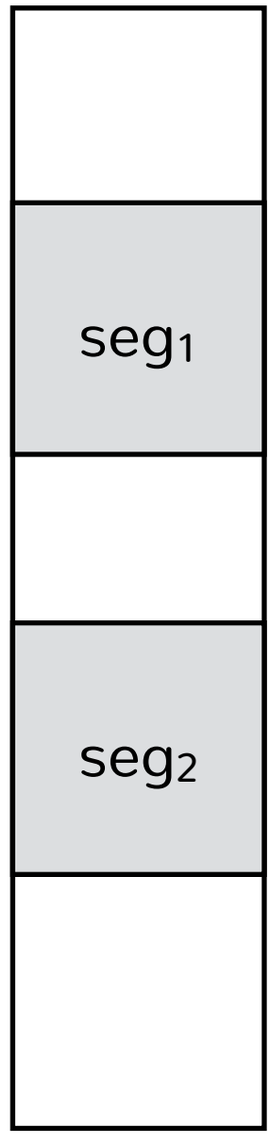
How does it provide C & I?

- Rewrite indirect jump, load, and store
- Segment matching approach
  - Upside: can pinpoint offending instruction
  - Downside?
- Address sandboxing approach
  - Mask upper bits of target address
  - Cost?



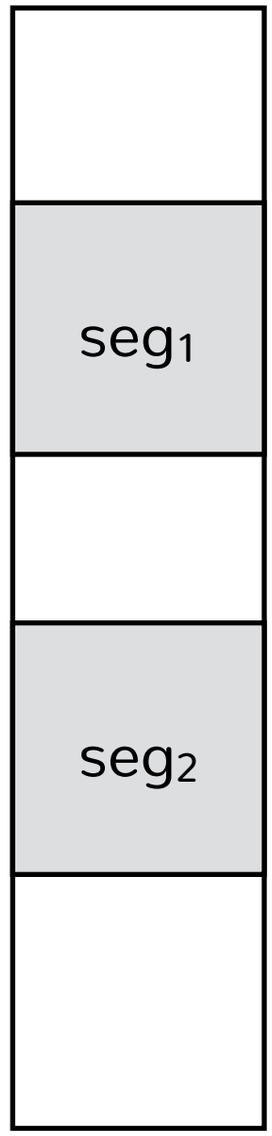
How does it provide C & I?

- Rewrite indirect jump, load, and store
- Segment matching approach
  - Upside: can pinpoint offending instruction
  - Downside? Performance!
- Address sandboxing approach
  - Mask upper bits of target address
  - Cost?



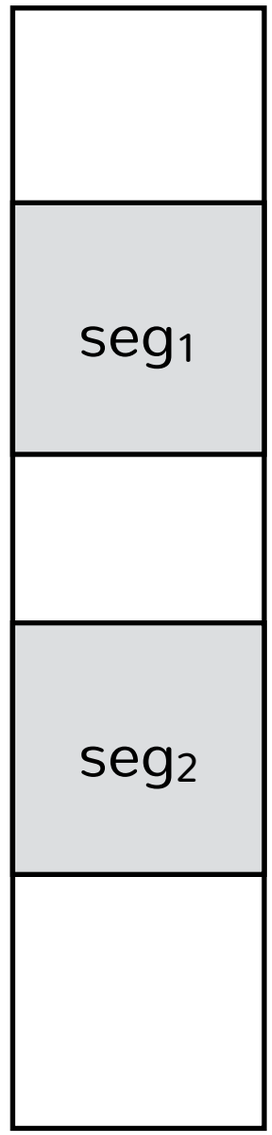
How does it provide C & I?

- Rewrite indirect jump, load, and store
- Segment matching approach
  - Upside: can pinpoint offending instruction
  - Downside? Performance!
- Address sandboxing approach
  - Mask upper bits of target address
  - Cost? 2 instructions per store + dedicated registers



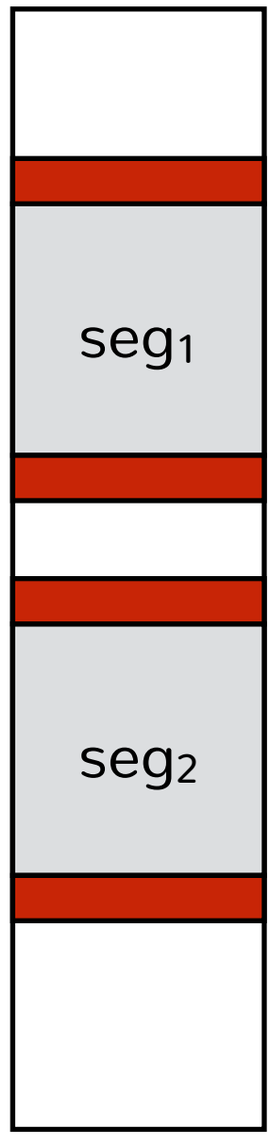
How does it provide C & I?

- Optimized address sandboxing approach
  - Use register-plus-offset instruction mode
- What do we need for this to work?



How does it provide C & I?

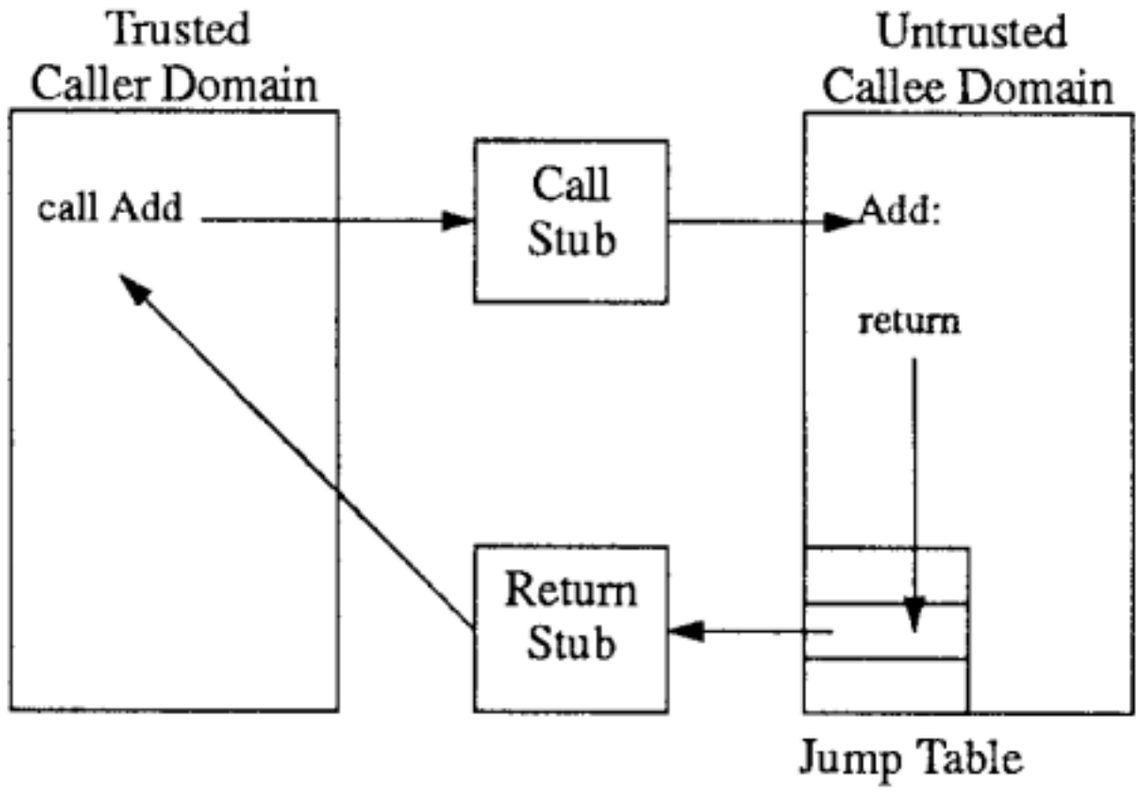
- Optimized address sandboxing approach
  - Use register-plus-offset instruction mode
- What do we need for this to work?



Are we done?

- A: yes, B: no

Need to mediate syscalls



This is super hard to get right in practice!

Google's Native Client

C1	Once loaded into the memory, the binary is not writable, enforced by OS-level protection mechanisms during execution.
C2	The binary is statically linked at a start address of zero, with the first byte of text at 64K.
C3	All indirect control transfers use a `nacljmp` pseudo-instruction (defined below).
C4	The binary is padded up to the nearest page with at least one `hlt` instruction (0xf4).
C5	The binary contains no instructions or pseudo-instructions overlapping a 32-byte boundary.
C6	All *valid* instruction addresses are reachable by a fall-through disassembly that starts at the load (base) address.
C7	All direct control transfers target valid instructions.

Table 1: Constraints for NaCl binaries.

Summary

- Secure design principles
  - Least privilege + privilege separation + isolation
- Different ways to do this with diff tradeoffs:
  - Use UIDs + namespaces + seccomp-bpf
  - Use syscall interposition
  - Use software-fault isolatoin