Chrome OS Internals

Josh Triplett
josh@joshtriplett.org

LinuxCon Europe 2014

Overview

- Intro to Chrome OS
- Architecture of Chrome OS
- Verified boot and developer mode
- Security
- Build a bootable Chromium OS image from source
- Develop Chrome OS

Chrome OS

- Operating system from Google based on the Chrome browser
- Designed around web apps
- Browser, Gmail, Google Docs, YouTube, Netflix, games
- Google Drive, Chrome Sync, and persistent app state
- Synced, backed up, and updated automatically

Chromium OS and Chrome OS

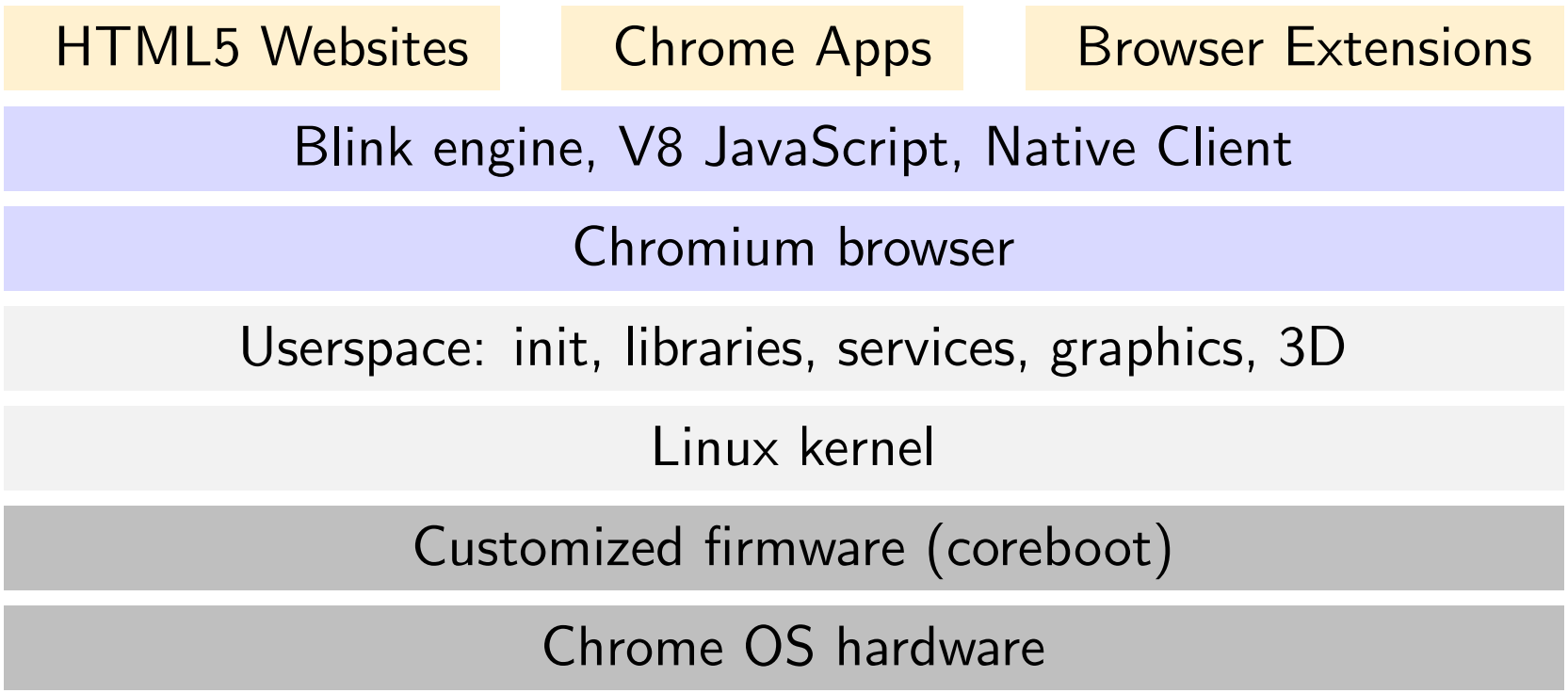
- Built from publically available Open Source code
- Only runs on devices in developer mode
- Allows shell and root access
- No Flash, Netflix, DRM

Chromium OS and Chrome OS

- Built from publically available Open Source code
- Only runs on devices in developer mode
- Allows shell and root access
- No Flash, Netflix, DRM

- Digital signature from Google
- Runs on systems in production mode
- Branding
- Flash, Netflix, and DRM

Architecture



Chrome OS Hardware

- Chromebook laptops
- Chromebox desktops
- Chromebase “all-in-ones” (built into a monitor)

Chrome OS Hardware

- Chromebook laptops
- Chromebox desktops
- Chromebase “all-in-ones” (built into a monitor)
- Arbitrary Linux-compatible PC hardware
  - Always effectively in developer mode

Hardware codenames

- Popular video game series for each hardware family
- Character for each model in that family

Hardware codenames

- Popular video game series for each hardware family
- Character for each model in that family
- Haswell: Star Fox
  - fox, slippy, falco, peppy

Hardware codenames

- Popular video game series for each hardware family
- Character for each model in that family
- Haswell: Star Fox
  - fox, slippy, falco, peppy
- Baytrail: Donkey Kong
  - rambi, squawks, quawks, swanky

Key differences in Chrome OS hardware

- Developer-mode switch (physical or keyboard-based)
- Custom keyboard and keyboard controller
- Hardware on Google compatibility list
- Embedded controller with Open Source firmware
- Uses coreboot-based Chrome OS firmware

Chrome OS firmware

- Based on coreboot and u-boot

Chrome OS firmware

- Based on coreboot and u-boot
- Coreboot provides the framework for hardware initialization
- “depthcharge”: u-boot as coreboot payload
  - Provides flexible boot of Linux from various media

Chrome OS firmware

- Based on coreboot and u-boot
- Coreboot provides the framework for hardware initialization
- "depthcharge": u-boot as coreboot payload
  - Provides flexible boot of Linux from various media
- Read-only firmware for root of trust and recovery mode
- A/B read-write firmware available for fallbacks during updates
- Includes SeaBIOS to boot arbitrary OSes

Chrome OS firmware

- Based on coreboot and u-boot
- Coreboot provides the framework for hardware initialization
- "depthcharge": u-boot as coreboot payload
  - Provides flexible boot of Linux from various media
- Read-only firmware for root of trust and recovery mode
- A/B read-write firmware available for fallbacks during updates
- Includes SeaBIOS to boot arbitrary OSes
- Open Source firmware for embedded controller

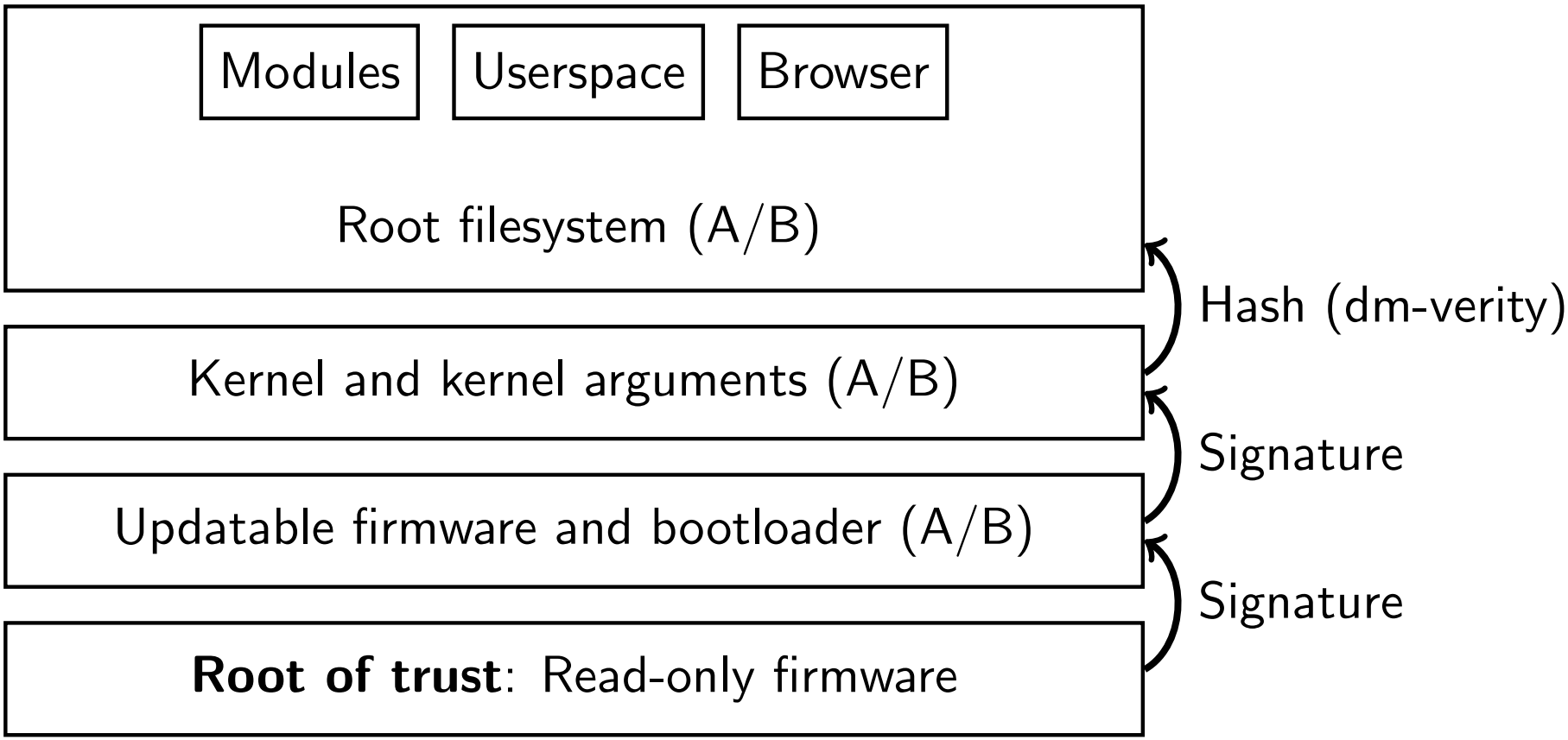
Chrome OS firmware

- Based on coreboot and u-boot
- Coreboot provides the framework for hardware initialization
- "depthcharge": u-boot as coreboot payload
  - Provides flexible boot of Linux from various media
- Read-only firmware for root of trust and recovery mode
- A/B read-write firmware available for fallbacks during updates
- Includes SeaBIOS to boot arbitrary OSes
- Open Source firmware for embedded controller
- Implements verified boot procedure

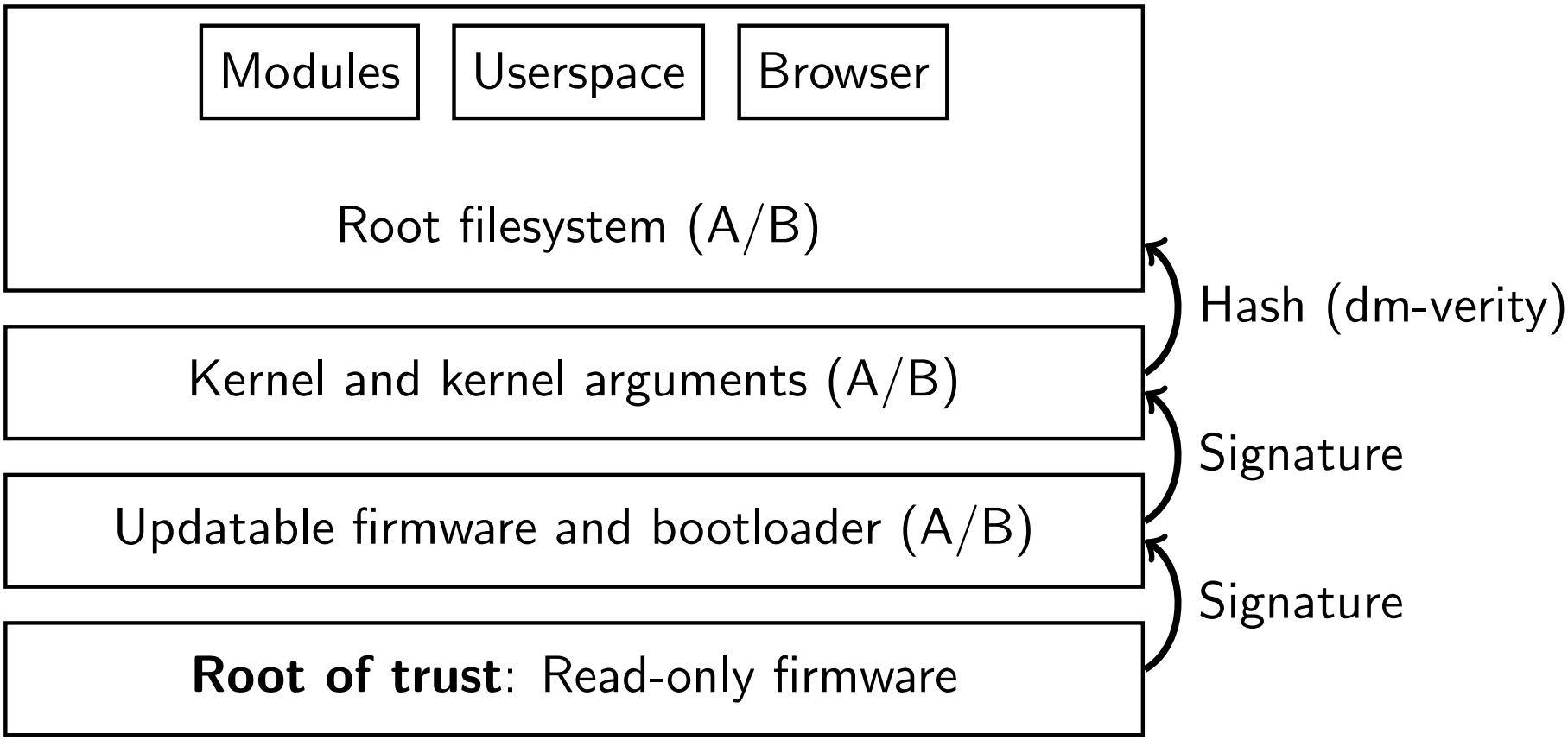
Chrome OS firmware

- Based on coreboot and u-boot
- Coreboot provides the framework for hardware initialization
- "depthcharge": u-boot as coreboot payload
  - Provides flexible boot of Linux from various media
- Read-only firmware for root of trust and recovery mode
- A/B read-write firmware available for fallbacks during updates
- Includes SeaBIOS to boot arbitrary OSes
- Open Source firmware for embedded controller
- Implements verified boot procedure
- Enforces developer-mode switch requirements
  - Physical presence (switch or keyboard)
  - Wiping local state when switching

Verified Boot



Verified Boot



All third-party code runs in a sandbox.

Developer mode

- Physical switch on older hardware
- Esc-Refresh-Power on newer hardware

Developer mode

- Physical switch on older hardware
- Esc-Refresh-Power on newer hardware
  - Tip: Refresh-Power is instant hard reset

Developer mode

- Physical switch on older hardware
- Esc-Refresh-Power on newer hardware
  - Tip: Refresh-Power is instant hard reset
- Allows bypassing verified boot via explicit keyboard interaction
- Enforced in firmware or embedded controller
- Not changeable from OS

Developer mode

- Physical switch on older hardware
- Esc-Refresh-Power on newer hardware
  - Tip: Refresh-Power is instant hard reset
- Allows bypassing verified boot via explicit keyboard interaction
- Enforced in firmware or embedded controller
- Not changeable from OS
- Wipes stateful partition, after enforced delay

Developer mode

- Physical switch on older hardware
- Esc-Refresh-Power on newer hardware
  - Tip: Refresh-Power is instant hard reset
- Allows bypassing verified boot via explicit keyboard interaction
- Enforced in firmware or embedded controller
- Not changeable from OS
- Wipes stateful partition, after enforced delay
- Allows booting USB or BIOS

Updates

- Chrome OS downloads and installs signed updates from Google
- Includes new firmware, kernel, and OS root
- Chrome OS keeps an A and B firmware, kernel, and root filesystem
- Flag un-booted versions, fall back to previously successful version if new version fails

Chrome OS kernel

- Extensively patched Linux kernel

Chrome OS kernel

- Extensively patched Linux kernel
  - Backported drivers and improvements
  - Security enhancements and hardening
  - Not new APIs

Chrome OS kernel

- Extensively patched Linux kernel
  - Backported drivers and improvements
  - Security enhancements and hardening
  - Not new APIs
- A/B copies for redundancy during updates

Chrome OS kernel

- Extensively patched Linux kernel
  - Backported drivers and improvements
  - Security enhancements and hardening
  - Not new APIs
- A/B copies for redundancy during updates
- Stored on dedicated partitions to simplify depthcharge
- Wrapped in verified boot container, with kernel command line
- Verification information for dm-verity on kernel command line
- Edit formatted kernel and command line via `vbutil_kernel`

Chrome OS userspace

- Linux distribution
- Based on Gentoo
  - `-O99 -funroll-loops -fomit-instructions -ftw`

Chrome OS userspace

- Linux distribution
- Based on Gentoo
  - `-O99 -funroll-loops -fomit-instructions -ftw`
- Uses the Portage build system and packaging infrastructure
- Pulls in many packages from Gentoo, and adds patches

Chrome OS userspace

- Linux distribution
- Based on Gentoo
  - `-O99 -funroll-loops -fomit-instructions -ftw`
- Uses the Portage build system and packaging infrastructure
- Pulls in many packages from Gentoo, and adds patches
- Adds its own chromiumos-overlay with the Chrome OS core and additional packages
- Adds board-specific overlay for each target board

Chrome OS userspace

- Linux distribution
- Based on Gentoo
  - `-O99 -funroll-loops -fomit-instructions -ftw`
- Uses the Portage build system and packaging infrastructure
- Pulls in many packages from Gentoo, and adds patches
- Adds its own chromiumos-overlay with the Chrome OS core and additional packages
- Adds board-specific overlay for each target board
- Notable divergence from Gentoo: Upstart

Chrome OS userspace stack

- Upstart and system daemons
- X Window System (for now)
- Mesa, libdrm, etc.
- Forks of ConnMan and ModemManager
- Custom audio server, cras
- Chrome browser, running Aura window manager
- Chrome browser windows

Chrome OS UI

- “Aura”
- Traditional window management
- Panel with fast-access app icons and app menu
- System tray, clock, notifications
- Designed with the Chrome OS keyboard in mind
- Runs in Chrome itself

Chrome OS UI

- “Aura”
- Traditional window management
- Panel with fast-access app icons and app menu
- System tray, clock, notifications
- Designed with the Chrome OS keyboard in mind
- Runs in Chrome itself
- X, Ozone, Freon

Chrome OS graphics

- Chrome GPU sandbox links to Mesa
  - Runs on X or GBM
  - Talks to graphics hardware
  - `/dev/dri/card0`
- GPU sandbox provides virtual GLES contexts
  - Validated
  - Isolated
- Browser engine, WebGL, and NaCl each get a GLES context
  - Communicate with GPU sandbox via command buffer

Chrome browser

- Almost all system components exist to support the browser
- Shares significant code with Chrome for Linux, but separate target
- Many different sandboxes
- Supports HTML5 and JavaScript with additional APIs
- Supports applications and extensions written in JavaScript
  - `https://developer.chrome.com/apps/api_index`
  - `https://developer.chrome.com/extensions/api_index`
- Supports native code via Native Client (NaCl)
  - `https://developer.chrome.com/native-client/pepper_dev/c`
  - Can port code from other platforms

Chrome Web Store

- Chrome OS's "app store"
- Most apps run on Chrome for Windows, Linux, or Chrome OS
- Apps runnable via system menu
- Apps and app data synced between Chrome browsers
- App format: .crx , a modified .zip
  - Same package used for all platforms
  - Prepended header includes signature via RSA and SHA-1
  - For more information:
    `https://developer.chrome.com/extensions/crx`

Native Client

- Sandboxed native code execution
- Uses seccomp BPF
- Based on Linux ELF file format
- C toolchain based on GCC and newlib or glibc
- Support for non-C languages
- Extensive Chrome-specific API
- Completely event driven; main thread may not block
- Ports of numerous major POSIX libraries

Security

Chrome OS threat model

- root $\neq$ kernel
- Enable local developers
- Protect against malware, especially persistent malware
- Protect against theft
- Slow down local attacks
- Defense in depth

Chrome OS security

- Extensive kernel and userspace hardening

Chrome OS security

- Extensive kernel and userspace hardening
- Verified boot, developer mode, and stateful wipe

Chrome OS security

- Extensive kernel and userspace hardening
- Verified boot, developer mode, and stateful wipe
- Per-user and per-system encrypted partitions (uses TPM, eCryptFS)

Chrome OS security

- Extensive kernel and userspace hardening
- Verified boot, developer mode, and stateful wipe
- Per-user and per-system encrypted partitions (uses TPM, eCryptFS)
- namespaces

Chrome OS security

- Extensive kernel and userspace hardening
- Verified boot, developer mode, and stateful wipe
- Per-user and per-system encrypted partitions (uses TPM, eCryptFS)
- namespaces
- seccomp

Chrome OS security

- Extensive kernel and userspace hardening
- Verified boot, developer mode, and stateful wipe
- Per-user and per-system encrypted partitions (uses TPM, eCryptFS)
- namespaces
- seccomp
- Most daemons run via “minijail”

Chrome OS security

- Extensive kernel and userspace hardening
- Verified boot, developer mode, and stateful wipe
- Per-user and per-system encrypted partitions (uses TPM, eCryptFS)
- namespaces
- seccomp
- Most daemons run via “minijail”
- No installable OS components or packages
  - Only changes via Chrome OS updates
  - Browser sandboxed

Additional hardening measures

- ASLR, user and kernel
- Hiding kernel pointers

Additional hardening measures

- ASLR, user and kernel
- Hiding kernel pointers
- Compiler hardening, including stack protection
- glibc checks

Additional hardening measures

- ASLR, user and kernel
- Hiding kernel pointers
- Compiler hardening, including stack protection
- glibc checks
- Restricted kernel-module loading

Additional hardening measures

- ASLR, user and kernel
- Hiding kernel pointers
- Compiler hardening, including stack protection
- glibc checks
- Restricted kernel-module loading
- Restricted device permissions and capabilities

Additional hardening measures

- ASLR, user and kernel
- Hiding kernel pointers
- Compiler hardening, including stack protection
- glibc checks
- Restricted kernel-module loading
- Restricted device permissions and capabilities
- Compiled out unnecessary security-sensitive components

Security policy

With a normal Chrome OS image, and developer mode off, it should not be possible to run any user-supplied native Linux executable or script.

User separation

- Chrome OS supports multiple users, and a “guest”
- Users tied to Google accounts
- Accounts theoretically identical across devices
- Each account has its own data, apps, etc

User separation

- Chrome OS supports multiple users, and a “guest”
- Users tied to Google accounts
- Accounts theoretically identical across devices
- Each account has its own data, apps, etc
- Accounts share networking and other system resources
  - Results in some confusing issues: need network to log in, and want to share networks among users, but cannot allow users to control the network used to log in.

Chrome browser security

- JavaScript sandboxing

Chrome browser security

- JavaScript sandboxing
- Native Client sandboxing
  - Code verification and analysis
  - Effectively native speed

Chrome browser security

- JavaScript sandboxing
- Native Client sandboxing
  - Code verification and analysis
  - Effectively native speed
- Tabs in separate, locked-down processes
- Media decoding and graphics in separate, locked-down processes

Chrome browser security

- JavaScript sandboxing
- Native Client sandboxing
  - Code verification and analysis
  - Effectively native speed
- Tabs in separate, locked-down processes
- Media decoding and graphics in separate, locked-down processes
- Sandboxed processes use seccomp BPF for syscall filtering

Chrome browser security

- JavaScript sandboxing
- Native Client sandboxing
  - Code verification and analysis
  - Effectively native speed
- Tabs in separate, locked-down processes
- Media decoding and graphics in separate, locked-down processes
- Sandboxed processes use seccomp BPF for syscall filtering
- Many features used opportunistically on Linux exist unconditionally on Chrome OS

Building

Getting Chrome OS Source

- Most of Chrome OS is tracked via git

Getting Chrome OS Source

- Most of Chrome OS is tracked via git
- A whole lot of git
  - Hundreds of repositories
  - Specific directory layout

Getting Chrome OS Source

- Most of Chrome OS is tracked via git
- A whole lot of git
  - Hundreds of repositories
  - Specific directory layout
- repo

Getting Chrome OS Source

- Most of Chrome OS is tracked via git
- A whole lot of git
  - Hundreds of repositories
  - Specific directory layout
- repo
- repo init -u $manifest_url
- repo sync

Getting Chrome OS Source

- Most of Chrome OS is tracked via git
- A whole lot of git
  - Hundreds of repositories
  - Specific directory layout
- repo
- repo init -u $manifest_url
- repo sync
- repo start
- repo upload

Bootstrapping via chroot

- Self-hosted build environment
- Avoids reliance on host tools and distribution

Bootstrapping via chroot

- Self-hosted build environment
- Avoids reliance on host tools and distribution
- `depot_tools`

Bootstrapping via chroot

- Self-hosted build environment
- Avoids reliance on host tools and distribution
- `depot_tools`
- `cros_sdk`

Bootstrapping via chroot

- Self-hosted build environment
- Avoids reliance on host tools and distribution
- `depot_tools`
- `cros_sdk`
  - Downloads initial binary chroot
  - Can rebuild from source

Bootstrapping via chroot

- Self-hosted build environment
- Avoids reliance on host tools and distribution
- `depot_tools`
- `cros_sdk`
  - Downloads initial binary chroot
  - Can rebuild from source
  - namespaces

Bootstrapping via chroot

- Self-hosted build environment
- Avoids reliance on host tools and distribution
- `depot_tools`
- `cros_sdk`
  - Downloads initial binary chroot
  - Can rebuild from source
  - namespaces
- Can run shell in chroot or act as command prefix
  - `cros_sdk --nousepkg -- build_command`
- Mounts source tree as `$HOME/trunk` in chroot

setup_board

- Set up build environment for each new target board
- Hardware codenames as mentioned earlier
- Generic target boards: amd64-generic, x86-generic
- Based on overlays in `src/overlays`

setup_board

- Set up build environment for each new target board
- Hardware codenames as mentioned earlier
- Generic target boards: amd64-generic, x86-generic
- Based on overlays in `src/overlays`
- `cros_sdk --nousepkg -- ./setup_board --board=$BOARD`

build_packages

- Build Gentoo packages from source
- Save the resulting binary packages

build_packages

- Build Gentoo packages from source
- Save the resulting binary packages
- `cros_sdk --nousepkg -- \`
  `./build_packages --board=$BOARD --nousepkg`

build_image

- Create root filesystem
- Install compiled binary packages onto it
- Construct disk image

build_image

- Create root filesystem
- Install compiled binary packages onto it
- Construct disk image
- ```
  cros_sdk --nousepkg -- \
  ./build_image --board=$BOARD \
  --noenable_rootfs_verification dev
  ```
  ```

build_image

- Create root filesystem
- Install compiled binary packages onto it
- Construct disk image
- `cros_sdk --nousepkg -- \`
  `./build_image --board=$BOARD \`
  `--noenable_rootfs_verification dev`
- base, dev, test

build_image

- Create root filesystem
- Install compiled binary packages onto it
- Construct disk image
- `cros_sdk --nousepkg -- \`
  `./build_image --board=$BOARD \`
  `--noenable_rootfs_verification dev`
- base, dev, test
- Based on metapackages in `src/third_party/chromiumos-overlay/chromeos-base`

rootfs verification

- Linux verifies root filesystem with dm-verity
- Mounting root read-write will break the hash

rootfs verification

- Linux verifies root filesystem with dm-verity
- Mounting root read-write will break the hash
- ext4 feature flags

rootfs verification

- Linux verifies root filesystem with dm-verity
- Mounting root read-write will break the hash
- ext4 feature flags
- Disable at build time with `--noenable_rootfs_verification`

rootfs verification

- Linux verifies root filesystem with dm-verity
- Mounting root read-write will break the hash
- ext4 feature flags
- Disable at build time with `--noenable_rootfs_verification`
- Disable on existing image with `/usr/share/vboot/bin/make_dev_ssh.sh --remove_rootfs_verification`

Image format

- GPT with 12 partitions
  - “Stateful” read-write partition (expands to disk size)
  - Linux kernel with header (A, B, and C)
  - Root filesystem (A, B, and C)
  - OEM
  - three reserved
  - EFI System Partition
- Bootable via coreboot/depthcharge, MBR (syslinux), and EFI (grub2)

Booting

- `./image_to_usb.sh`
- `./image_to_vm.sh`

Developing

Chrome OS development

- Uses repo to manage several hundred git repositories

Chrome OS development

- Uses repo to manage several hundred git repositories
  - repo start, repo upload

Chrome OS development

- Uses repo to manage several hundred git repositories
  - repo start, repo upload
- Uses gerrit to accept and review contributions

Chrome OS development

- Uses repo to manage several hundred git repositories
  - repo start, repo upload
- Uses gerrit to accept and review contributions
- All changes require code review before merging

Chrome OS development

- Uses repo to manage several hundred git repositories
  - repo start, repo upload
- Uses gerrit to accept and review contributions
- All changes require code review before merging
- Changes built and tested on numerous Chrome OS platforms before merging
- Continous integration via buildbot

Developing the Chrome browser

- Download source separately

Developing the Chrome browser

- Download source separately
- Similar multi-repository structure
- Uses gclient in place of repo
- Uses reitveld in place of gerrit
- (Both support subversion in addition to git)

Developing the Chrome browser

- Download source separately
- Similar multi-repository structure
- Uses gclient in place of repo
- Uses reitveld in place of gerrit
- (Both support subversion in addition to git)
- `chromeos-base/chromeos-chrome`

Developing the Chrome browser

- Download source separately
- Similar multi-repository structure
- Uses gclient in place of repo
- Uses reitveld in place of gerrit
- (Both support subversion in addition to git)
- `chromeos-base/chromeos-chrome`
- `CHROME_ORIGIN=LOCAL_SOURCE`

Modifying packages

- ebuild
- `src/third_party/chromiumos-overlay`

Modifying packages

- ebuild
- `src/third_party/chromiumos-overlay`
- Extensive use of eclass

Modifying packages

- ebuild
- `src/third_party/chromiumos-overlay`
- Extensive use of eclass
- No universal approach for package modification
- Many common patterns

Modifying packages

- ebuild
- `src/third_party/chromiumos-overlay`
- Extensive use of eclass
- No universal approach for package modification
- Many common patterns
- Some packages download tarballs and apply patches
- Some packages clone git repositories (and apply patches)
- Some packages use `cros_workon`

cros_workon

- ebuild uses `cros_workon` eclass

cros_workon

- ebuild uses `cros_workon` eclass
- ebuild references existing checked-out git repository (from repo)
- ebuild specifies git commit and tree hashes
- Normal build checks out and builds that commit

cros_workon

- ebuild uses `cros_workon` eclass
- ebuild references existing checked-out git repository (from repo)
- ebuild specifies git commit and tree hashes
- Normal build checks out and builds that commit
- `cros_workon start` unmasks ebuild version 9999
- 9999 ebuild builds the checked-out version (including local changes)

Package management

- Portage tools provides for host and each board
  - emerge, equery: for the host chroot
  - `emerge-${BOARD}`, `equery-${BOARD}`: for target board
- Used during `build_packages` and `build_image`

Package management

- Portage tools provides for host and each board
  - emerge, equery: for the host chroot
  - `emerge-${BOARD}`, `equery-${BOARD}`: for target board
- Used during `build_packages` and `build_image`
- Can install individual packages in developer mode
- Use `emerge-${BOARD}` to build
- Use `cros deploy` (formerly `gmerge`) to remotely deploy

Come work on Chrome OS!

`https://01.org/jobs`

Come work on Chrome OS!

`https://01.org/jobs`

Questions?