GOOGLE CHROME OS

A Technical Seminar Report Submitted to

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, HYDERABAD In

Partial Fulfillment of the requirement For the Award of the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

Undru Harsha Vardhan **(H.T.N0:19N01A05A8)**

Under the Supervision of

Mr.M.Murali Mohan Reddy

Assistant Professor

Department of Computer Science and Engineering

SREE CHAITANYA COLLEGE OF ENGINEERING

(Affiliated to JNTUH, HYDERABAD)

THIMMAPUR, KARIMNAGAR, TELANGANA-505 527

NOVEMBER-2022

SREE CHAITANYA COLLEGE OF ENGINEERING

(Affiliated to JNTUH, HYDERABAD)

THIMMAPUR, KARIMNAGAR, TELANGANA-505 527

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Technical Seminar report entitled **"Google Chrome OS"** is being submitted by **Undru Harsha Vardhan** bearing hall ticket number: **19N01A05A8** for partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** discipline to the **Jawaharlal Nehru Technological University, Hyderabad** during the academic year 2022 - 2023 is a bonafide work carried out by him under my guidance and supervision.

The result embodied in this report has not been submitted to any other University or institution for the award of any degree of diploma.

Guide

Mr.M.Murali Mohan Reddy
Assistant Professor
Department of CSE

Head of the Department

Mr.Khaja Ziauddin
Associate Professor
Department of CSE

SREE CHAITANYA COLLEGE OF ENGINEERING

(Affiliated to JNTUH, HYDERABAD)

THIMMAPUR, KARIMNAGAR, TELANGANA-505 527

Department of Computer Science and Engineering

DECLARATION

I, Undru Harsha Vardhan, is student of **Bachelor of Technology in Computer Science and Engineering,** during the academic year: 2022-2023**,** hereby declare that the work presented in this Technical Seminar entitled **Google Chrome OS** is the outcome of our own bona fide work and is correct to the best of our knowledge and this work has been undertaken taking care of Engineering Ethics and carried out under the supervision of **Mr.M.Murali Mohan Reddy ,Assistant Professor**.

It contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Undru Harsha Vardhan(H.T.N0:19N01A05A8)

Date:

Place

SREE CHAITANYA COLLEGE OF ENGINEERING
(Affiliated to JNTUH, HYDERABAD)
THIMMAPUR, KARIMNAGAR, TELANGANA-505 527
Department of Computer Science and Engineering

ACKNOWLEDGEMENTS

The Satisfaction that accomplishes the successful completion of any task would be incomplete without the mention of the people who make it possible and whose constant guidance and encouragement crown all the effortswith success.

I would like to express my sincere gratitude and indebtedness to my seminar supervisor ***M.Murali Mohan Reddy, Assistant Professor***, Department of Computer Science and Engineering, Sree Chaitanya College of Engineering, LMD Colony, Karimnagar for his/her valuable suggestions and interest throughout the course of this project

I am also thankful to Head of the department ***Mr.*Khaja Ziauddin *Associate Professor & HOD***, Department of Computer Science and Engineering, Sree Chaitanya College of Engineering,LMD Colony,Karimnagar for providing excellent infrastructure and a nice atmosphere for completing this project successfully

We Sincerely extend out thanks to ***Dr.G.Venkateswarlu,Principal,*** Sree Chaitanya College of Engineering,LMD Colony,Karimnagar,for providing all the facilities required for completion of this Technical Seminar.

I convey my heartfelt thanks to the lab staff for allowing me to use the required equipment whenever needed. Finally, I would like to take this opportunity to thank my family for their support through the work.

I sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of thiswork.

Undru Harsha Vardhan.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Google Chrome OS is a Linux operating system designed by Google to work exclusively with web applications. It is intended to focus on Web applications while running a fast and simple interface, based off Google's existing Chrome browser. Google announced the operating system on July 7, 2009 and made it an open source project, called Chromium OS, that November.Unlike Chromium OS, which can be compiled from the downloaded source code, Chrome OS will only ship on specific hardware from Google's manufacturing partners. The user interface takes a minimalist approach, resembling that of the Chrome web browser. Because Google Chrome OS is aimed at users who spend most of their computer time on the Internet, the only application on the device will be a browser incorporating a media player.Google Chrome OS is initially intended for secondary devices like netbooks, not as a user's primary PC, and will run on hardware incorporating an x86 or ARM-based processor. Chrome OS as a "hardened" operating system featuring auto-updating and sandbox features that will reduce malware exposure. Google claimed that Chrome OS would be the most secure consumer operating system due in part to a verified boot capability, in which the initial boot code, stored in read-only memory, checks for system compromises.

1 INTRODUCTION

1.1Chromium OS

Chromium OS is an open-source project that aims to build an operating system that provides a fast, simple, and more secure computing experience for people who spend most of their time on the web.

1.2History

Google developers began coding the operating system in 2009, inspired by the growing popularity and lower-power consumption of netbooks, and the realization that these small laptops had gotten their name from their primary use: accessing the Internet. To ascertain demand for an operating system focused on netbook Web transactions, the company eschewed the usual demographic research generally associated with a large software development project. Instead, engineers have relied on more informal metrics, including monitoring the usage patterns of some 200 Chrome OS machines used by Google employees. Developers also noted their own usage patterns. Matthew Papakipos, engineering director for the Chrome OS project, put three machines in his house and found himself logging in for brief sessions: to make a single search query or send a short email.

On November 19, 2009, Google released Chrome OS's source code under the BSD license as the Chromium OS project. As with other open source projects, developers are modifying code from Chromium OS and building their own versions, whereas Google Chrome OS code will only be supported by Google and its partners, and will only run on hardware designed for the purpose. Unlike Chromium OS, Chrome OS will be automatically updated to the latest version. InformationWeek reviewer Serdar Yegulalp wrote that Chrome OS will be a product, developed to "a level of polish and a degree of integration with its host hardware that Chromium OS does not have by default," whereas Chromium OS is a project, "a common baseline from which the finished work is derived"

as well a pool for derivative works. The product and project will be developed in parallel and borrow from each other.

1.3User interface

Design goals for Google Chrome OS's user interface include using minimal screen space by combining applications and standard Web pages into a single tab strip, rather than separating the two. Designers are considering a reduced window management scheme that would operate only in full-screen mode. Secondary tasks would be handled with "panels": floating windows that dock to the bottom of the screen for tasks like chat and music players. Split screens are also under consideration for viewing two pieces of content side-by-side. Google Chrome OS will follow the Chrome browser's practice of leveraging HTML5's offline modes, background processing, and notifications. Designers propose using search and pinned tabs as a way to quickly locate and access applications.

1.4 Architecture

In preliminary design documents for the Chromium OS open source project, Google describes a three-tier architecture: firmware, browser and window manager, and system-level software and userland services.

- The firmware contributes to fast boot time by not probing for hardware, such as floppy disk drives, that are no longer common on computers, especially netbooks. The firmware also contributes to security by verifying each step in the boot process and incorporating system recovery.
- System-level software includes the Linux kernel that has been patched to improve boot performance. Userland software has been trimmed to essentials, with management by Upstart, which can launch services in parallel, re-spawn crashed jobs, and defer services in the interest of faster booting.
- The window manager handles user interaction with multiple client windows much like other X window managers.

1.5 Hardware support

Google Chrome OS is initially intended for secondary devices like netbooks, not a user's primary PC, and will run on hardware incorporating an x86 or ARM. While Chrome OS will support hard disk drives, Google has requested that its hardware partners use solid-state drives due to their higher performance and reliability, as well as the lower capacity requirements inherent in an operating system that accesses applications and most user data on remote servers. Google Chrome OS consumes one-sixtieth as much drive space as Windows 7.

Companies developing hardware for the operating system include Hewlett-Packard, Acer, Adobe, Asus, Lenovo, Texas Instruments, Freescale, Intel, Samsung Australia and Qualcomm.

In December 2009, Michael Arrington of TechCrunch reported that Google has approached at least one hardware manufacturer about building a Google-branded Chrome OS netbook. According to Arrington's sources, the devices could possibly be configured for mobile broadband and be subsidized by one or more carriers.[

1.6 Integrated media player

Google will integrate a media player into both Chrome OS and the Chrome browser, enabling users to play back MP3s, view JPEGs, and handle other multimedia files while offline.

1.7 Link handling

One unresolved design problem related to both Chrome OS and the Chrome browser is the desired behavior for how Web applications handle specific link types. For example, if a JPEG is opened in Chrome or on a Chrome OS device, should a specific Web application be automatically opened to view it, and if so, which one? Similarly, if a user clicks on a .doc file, which website should open: Office Live, Gview, or a previewing utility? Project director Matthew Papakipos noted that Windows developers

have faced the same fundamental problem: "Quicktime is fighting with Windows Media Player, which is fighting with Chrome". As the number of Web applications increases, the same problem arises.

1.8 Market implications

When Google announced the Chrome browser in September 2008 it was viewed as a continuation of the battle between Google and Microsoft ("the two giants of the digital revolution"). As of December 2009, Microsoft dominates the usage share of desktop operating systems and the software market in word processing and spreadsheet applications. The operating system dominance may be challenged directly by Google Chrome OS, and the application dominance indirectly through a shift to cloud computing. According to an analysis by PC World, Google Chrome OS represents the next step in this battle. But Chrome OS engineering director Matthew Papakipos has noted that the two operating systems will not fully overlap in functionality. Users should be aware that Chrome OS hosted on a netbook is not intended as a substitute for Microsoft Windows running on a conventional laptop, which has the computational power to run a resource-intensive program like Photoshop.

In November 2009, Glyn Moody, writing for *Linux Journal*, predicted that Google's market model for the Chrome OS will be to give the software and the netbook hardware that it will run on away for free, as a means of expanding its advertising-based model. He said: "The unexpected success of netbooks over the last two years shows there is a market for this new kind of computing; giving away systems for free would take it to the next level. Then, gradually, that instant-on, secure, secondary netbook might become the one you spend most time on, and Google's ad revenues would climb even higher...."

1.9 Relationship to Android

The successive introductions of Android and Google Chrome OS, both open source, client-based operating systems, have created some market confusion, especially with Android's growing success. Microsoft CEO Steve Ballmer accused Google of not

being able to make up their mind.[Google has downplayed this conflict, suggesting that the two operating systems address different markets, mobile and personal computing, which remain distinct despite the growing convergence of the devices. Co-founder Sergey Brin suggested that the two systems "will likely converge over time".

2 Motivation and background:

Google designed Chrome for people who live on the web — searching for information, checking email, catching up on the news, shopping or just staying in touch with friends. However, the operating systems that browsers run on were designed in an era where there was no web. So today, Google's announcing a new project that's a natural extension of Google Chrome — the Google Chrome Operating System. It's Google's attempt to re-think what operating systems should be.

Computers need to get better. People want to get to their email instantly, without wasting time waiting for their computers to boot and browsers to start up. They want their computers to always run as fast as when they first bought them. They want their data to be accessible to them wherever they are and not have to worry about losing their computer or forgetting to back up files. Even more importantly, they don't want to spend hours configuring their computers to work with every new piece of hardware, or have to worry about constant software updates. And any time our users have a better computing experience, Google benefits as well by having happier users who are more likely to spend time on the Internet.

3 Related Work:

Google Chrome OS is a new project, separate from Android. Android was designed from the beginning to work across a variety of devices from phones to set-top boxes to netbooks. Google Chrome OS is being created for people who spend most of their time on the web, and is being designed to power computers ranging from small netbooks to full-size desktop systems. While there are areas where Google Chrome OS and Android overlap, Google believes choice will drive innovation for the benefit of everyone, including Google.

Google Chrome OS is an open source, lightweight operating system that will initially be targeted at netbooks. Later this year Google will open-source its code, and netbooks running Google Chrome OS will be available for consumers in the second half of 2010. Because Google's already talking to partners about the project, and it'll soon be working with the open source community, Google wanted to share our vision now so everyone understands what we are trying to achieve.

Google has a lot of work to do, and Google’s definitely going to need a lot of help from the open source community to accomplish this vision.

4 Software Architecture:

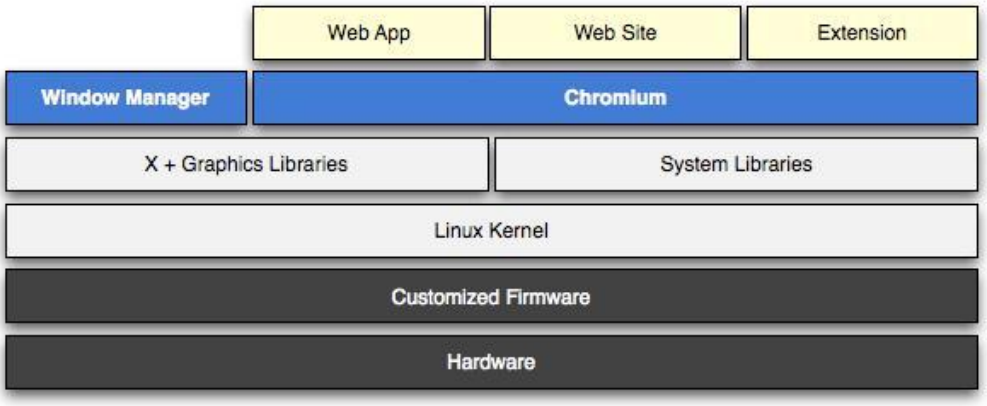

Fig 4.1 Software architecture

High-level design

We'll look at each component, starting with the firmware.

4.1 Firmware

The firmware plays a key part to make booting the OS faster and more secure. To achieve this goal we are removing unnecessary components and adding support for verifying each step in the boot process. We are also adding support for system recovery into the firmware itself. We can avoid the complexity that's in most PC firmware because we don't have to be backwards compatible with a large amount of legacy hardware. For example, we don't have to probe for floppy drives.

Our firmware will implement the following functionality:

- **System recovery:** The recovery firmware can re-install Chromium OS in the event that the system has become corrupt or compromised.
- **Verified boot:** Each time the system boots, Chromium OS verifies that the firmware, kernel, and system image have not been tampered with or become corrupt. This process starts in the firmware.
- **Fast boot:** We have improved boot performance by removing a lot of complexity that is normally found in PC firmware.

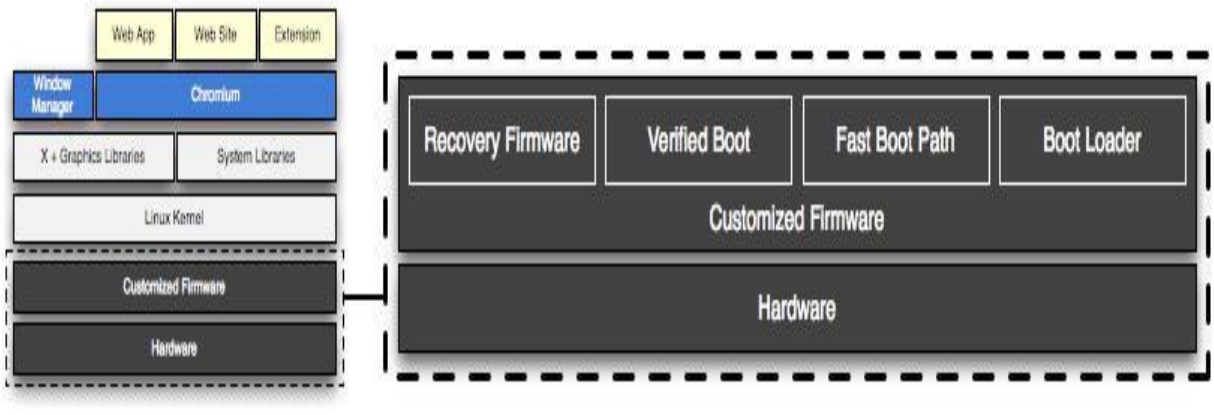

Fig 4.2 hardware and firmware

4.2 System-level and user-land software

From here we bring in the Linux kernel, drivers, and user-land daemons. Our kernel is mostly stock except for a handful of patches that we pull in to improve boot performance. On the user-land side of things we have streamlined the init process so that we're only running services that are critical. All of the user-land services are managed by Upstart. By using Upstart we are able to start services in parallel, re-spawn jobs that crash, and defer services to make boot faster.

Here's a quick list of things that we depend on:

- **D-Bus:** The browser uses D-Bus to interact with the rest of the system. Examples of this include the battery meter and network picker.
- **Connection Manager:** Provides a common API for interacting with the network devices, provides a DNS proxy, and manages network services for 3G, wireless, and ethernet.
- **WPA Supplicant:** Used to connect to wireless networks.
- **Autoupdate:** Our autoupdate daemon silently installs new system images.
- **Power Management:** (ACPI on Intel) Handles power management events like closing the lid or pushing the power button.
- **xscreensaver:** Handles screen locking when the machine is idle.
- **Standard Linux services:** NTP, syslog, and cron.

4.3 Chromium and the window manager

The window manager is responsible for handling the user's interaction with multiple client windows. It does this in a manner similar to that of other X window managers, by controlling window placement, assigning the input focus, and exposing hotkeys that exist outside the scope of a single browser window. Parts of the ICCCM (Inter-Client Communication Conventions Manual) and EWHM (Extended Window Manager Hints) specifications are used for communication between clients and the window manager where possible.

The window manager also uses the XComposite extension to redirect client windows to offscreen pixmaps so that it can draw a final, composited image incorporating their contents itself. This lets windows be transformed and blended together. The Clutter library is currently used to animate these windows and to render them via OpenGL or OpenGL|ES.

5 Firmware Boot and Recovery

5.1 Firmware

1. Incomplete update: An update of the firmware is interrupted. This leaves the portion of the firmware which was being updated in an unknown or corrupt state. For example, if the update is interrupted after a firmware block is erased but before it is reprogrammed, that block is empty.
2. Attack: An attacker compromises the software and is able to reprogram the firmware. For example, an exploit of an unpatched kernel vulnerability. In this case, both the main and backup firmware may be compromised.
3. Corruption: The EEPROM holding the firmware becomes corrupted in the sectors containing writable/updatable firmware.

5.2 Software

1. Incomplete update: An update of the software on the drive is interrupted. This leaves the rootfs partition in an unknown state.
2. Attack: An attacker compromises the software and is able to rewrite the data on the drive (rootfs or partition table).
3. Malicious user: A malicious user installs developer mode onto the device, leaving behind a trojan, then returns the device.
4. Corruption: The drive becomes corrupted in the partition table or rootfs partition.
5. Crash: Device crashes on boot due to bad software. For example, the device is updated with the wrong image. This prevents the normal autoupdate process from running.

5.3 Design decisions

Boot needs to start securely

In order to attempt a secure boot, the initial boot stub needs to perform a minimum level of initialization to verify the next piece of boot code before handing off to that code.

To prevent accidental or intentional corruption of the known-good boot stub, this code must be in a portion of memory which is not field-writable. Many EEPROM devices have an external pin (WP) which can be pulled low to write protect the upper portion of the EEPROM. This has a number of benefits:

- Devices are writable at time of manufacture (as opposed to true ROMs, which are fixed at time of ROM manufacture).
- Devices can be made writable for firmware development by simple hardware modification.
- Both readable and read-only ROM are provided by a single device. Simpler board design, fewer parts, lower cost.
- Any attacker who can open the case and modify the hardware to write to the protected upper portion of ROM could also simply replace a true ROM with a reprogrammed part, so this isn't significantly less secure than a true ROM.

On ARM platforms, the inital boot ROM may be in the same package as the processor.

This is even more secure compared to a separate EEPROM. Writable firmware should have a backup

To protect against a failed firmware update, the writable portion of the firmware (responsible for doing the remainder of chipset and storage setup and then bootstrapping the kernel off the storage device) should exist in two copies. In the event the first copy is corrupt, the device can boot normally off the second copy. This is similar to the design for the file system, which has two copies of the root partition.

Recovery firmware must be read-only

Recovery firmware must be able to take over the boot process if the boot stub determines that the normal writable firmware is corrupt, or if the user manually boots the device into recovery mode.

To prevent the recovery firmware from being corrupted by a firmware update or a software-based attack, it must be in the same read-only portion of the EEPROM as the boot stub.

Recovery firmware does not need to access the network

The recover process should not require firmware-level network access by the device being recovered. The recovery procedure can involve a second computer, which is used to create recovery media (for example, a USB drive or SD card). It is assumed that second computer has network access.

This simplifies the recovery process, since the recovery firmware only needs to bring up enough of the system to bootstrap a Linux image on local storage. That image can then take care of reflashing the EEPROM and reimaging.

It is not necessary to implement a full network stack with WiFi configuration in the recovery firmware to support PXE boot. PXE boot introduces a number of complications:

- Need to initialize more hardware to bring up wireless, keyboard, etc.
- Need to implement a full network stack.
- Makes recovery an interactive process, including the user entering their SSID and WPA key, which the user may not know.
- Unlikely to work for public WiFi access points; these often redirect http access to a login screen, navigation of which which would necessitate a full browser in the recovery firmware.
- Unlikely to work for 3G, if that requires more complicated drivers or configuration.

All of these issues would need to be resolved, and the resulting firmware must be correct at the time the device ships, because recovery mode firmware can't be updated in the field. It is unacceptable to ship a mostly-working PXE solution, assuming that the user can fall back on a second computer in the event PXE recovery fails. The only time

the user would discover PXE recovery didn't work is when the user is relying on it to repair their computer.

Recovery firmware should tell the user how to recover

If the recovery firmware finds a USB drive/SD card with a good recovery image on it, it should boot it and use that to recover. The software in that recovery image will have its own user interface to guide the user through recovery.

If the recovery firmware does not find a good recovery image, it needs to tell the user how to use a second computer to build that recovery image.

The preferred way to do this is to initialize the screen and show recovery instructions to the user, including a URL to go to in that second computer's web browser. Note that recovery instructions need to be displayed in the correct language for the user.

It is desirable for the recovery instructions and/or recovery URL to include a code for the device model. This allows the destination website to:

- Provide the proper recovery image for that device model.
- Describe the recovery procedure specific to that device model. For example, if the device has a SD card but no USB port, the recovery procedure would indicate that a SD card is necessary, and would not discuss the possibility of recovering using USB.
- Display graphics appropriate for the device model. For example, showing the location of the USB or SD card port.

Users must be able to manually trigger recovery mode

If the writable firmware and/or rootfs have valid signatures but don't work (for example, the user somehow managed to get an ARM kernel on an x86 device), the user needs to be able to force recovery mode to run.

This can be done by having a physical reset button somewhere on the device. When this button is pressed during power-on, the device goes straight to the recovery firmware without even looking at the writable firmware or file system.

Some options for the recovery button:

- It could be a button attached to a GPIO on the main processor. In this case, the boot stub would initialize the GPIO and read its state at boot time.
- It could be a button attached to a subprocessor such as the Embedded Controller (EC). In this case, the boot stub would need to request the button state from the EC at boot time.
- It could be one of the keys on the keyboard, though this creates the undesirable possibility of accidentally entering recovery mode.
    - This is undesirable if the only interface to the keyboard is USB, because USB firmware is more complex and the USB hardware interface can take a significant amount of time to initialize.
    - Some devices use a subprocessor to read the keyboard. In this case, initiating recovery mode is much like the previous option.
    - The keyboard could bring out the press-state of one of its keys to a separate I/O line, which could be attached to a GPIO on the processor or to a subprocessor.

Since the recoverability of Chromium OS is one of its key features, we seek to have a dedicated "recovery mode" button.

Support developers / l33t users installing their own software

To provide a relatively trustable boot for the majority of users, we need to control all the read-only firmware at the beginning of the boot process.

To support developers, at some point during the boot process, we need to hand off to code self-signed by someone else.

The initial devices will allow handoff at the point the kernel is loaded and its embedded signature is checked. This can produce one of three results:

- Trusted kernel: The kernel has a valid signature, and the signing key is known to the firmware. This is the normal case for production devices.
- Developer kernel: The kernel has a valid signature, but the key used to sign the kernel is not known to the firmware. This is the case when a developer builds and self-signs their own kernel.
- Corrupt kernel: The kernel fails its signature check.

Once the chain of trust departs from the standard Chromium OS boot chain, we need to indicate this clearly to the user of the device. This prevents malicious attackers from giving users a modified version of Chromium OS without the user knowing. We likely will need to show a warning screen which includes the following elements:

- A warning that the standard image of Chromium OS is not running
- A means of reverting back to the standard Chromium OS image
- A means of allowing the user/developer to proceed down the "untrusted" path

It is desirable for the warning screen to have a timeout, so that Chromium OS devices with developer images can be used in unattended applications (for example, as a media server). The timeout should be sufficiently long that a user can read and respond to it - for example, at least 30 seconds.

Since language settings will not be available at this stage of the boot process, any messaging will likely need to be internationalized and displayed in all possible la

6 File System/Autoupdate

6.1 Goals

The autoupdate system has the following goals:

- Updates are delta-compressed over the wire.
- Delta updates are quick to apply.
- Root file system is mounted read-only (another mount point is read/write for storing state).
- Device can automatically revert to previous installed version of the OS if the newly installed OS fails to boot properly.
- Updates are automatic and silent.
- Updates should be small.
- System protects users from security exploits.
- System never requires more than one reboot for an update.
- There's a cap on how much space the root file system, previous version(s), downloaded updates, etc. can take up.
- There's a clear division between the storage locations on the drive for system software and for user data.
- The new update must be set bit-for-bit from the server, since we will be signing each physical block on the system partition. (For more information, see the Verified Boot design document.)

6.2 Partitions

A drive currently contains up to four partitions:

- One partition for state resident on the drive (user's home directory/Chromium profile, logs, etc.)—called the "stateful partition."
- An optional swap partition.
- Two partitions for the root file system.

In the future, drives may be able to have more partitions, as needed. Because we can use extended partitions, we aren't limited to four partitions.

6.3 Root file system

Only one of the two partitions designated for the root file system will be in use at a given time. The other will be used for autoupdating and for a fallback if the current partition fails to boot.

While a partition is in use as the boot partition, it's read-only until the next boot. Not even the autoupdater will edit the currently-booted root file system. We will mount the stateful partition read-write and use that for all state that needs to be stored locally.

During autoupdate, we will write to the other system partition. Only the updater (and apps running as root) will have access to that partition.

The update process

Open issue: How to efficiently deploy delta updates that result in a partition that's bit-for-bit what we want?

6.4 Successful boot

The update process relies partly on the concept of a "successful boot." At any given point, we will be able to say one of the following things:

- This system booted up correctly.
- This system booted up incorrectly.
- The system is currently attempting to boot, so we don't know yet.

Open issue: For a boot to count as successful, does the user have to successfully log in? Does the OS have to successfully ping the update server? Once a system has booted successfully, we consider the other root partition to be available for overwriting with an autoupdate.

6.5 Limiting the number of boot attempts

An updated partition can attempt to boot only a limited number of times; if it doesn't boot successfully after a couple of attempts, then the system goes back to booting from the other partition. The number of attempts is limited as follows:

When a partition has successfully been updated, it's assigned a remaining_attempts value, probably 1 or 2. This value will be stored in the partition table next to the bootable flag (there are some unused bits that the boot loader can use for its own purposes). The boot loader will examine all partitions in the system; if it finds any partition that has a remaining_attempts value > 0, it will decrement remaining_attempts and then attempt to boot from that partition. If the boot fails, then this process repeats.

If no partitions have a remaining_attempts value > 0, the boot loader will boot from a partition marked bootable, as a traditional boot loader would. Open issue: What's the initial value for remaining_attempts?

6.6 Boot Sequence

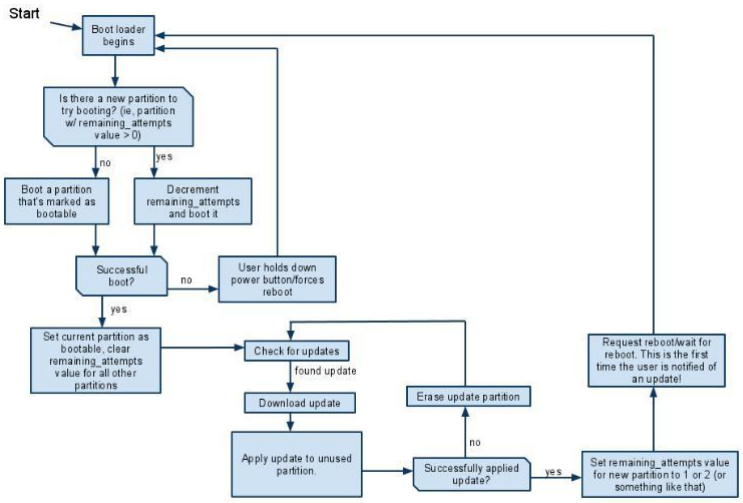

Fig 6.1 Boot sequence

6.7 Disk speed and battery life

Simple benchmarks indicate that dm-crypt can perform many operations at approximately the same speed and power cost as the system does without encryption. It's when sustained reads and writes occur that more and more CPU is used. A test with a heavy, sustained write resulted in the same battery discharge rate as the heavy writing used without encryption, but the encrypted large write took about twice as long to complete.

In most use cases, disk encryption isn't noticeable. If AES acceleration reaches additional processors, then the impact will be even lower. .

Directory structure

All metadata for this feature will live under the /home/.shadow directory. Each user will have a subdirectory with a name based on the user name hash. That directory will contain all data related to that user's image on the machine.For example:

/home/.shadow/da39a3ee5e6b4b0d3255bfef95601890afd80709/image
/home/.shadow/da39a3ee5e6b4b0d3255bfef95601890afd80709/salt.0
/home/.shadow/da39a3ee5e6b4b0d3255bfef95601890afd80709/key.0

7 Security Overview

7.1 Protecting Cached User Data

Problem: Multiple users, portable devices, locally stored data

Chromium OS devices should provide privacy protection for user data stored on the local disk. In particular, some subset of a user's email, pictures, and even HTTP cookies will be stored on the local drive to enhance the online experience and ensure Chromium OS devices are useful even when an Internet connection isn't available. Without this protection, anyone who has physical access to the Chromium OS device will be able to access any data cached on it.

Normally, data stored in the cloud is protected by privacy-guarding mechanisms provided by the servers that host it. When data is pulled out of the cloud, it moves outside of the reach of those server-provided protections. Even though this is the case today with most computers, there are two areas that make this even more important to Chromium OS devices:

- With a Chromium OS device, users shouldn't have to worry about whether other users of the same device can *see*their cached data. For example, if my friend wants to log in to my device to check her mail, it's not a big deal. She won't be able to access my data, and even though she used my device, I won't be able to access her data.
- Chromium OS is designed for portable devices. The risk with very portable devices is that they are easy to misplace or leave behind. We don't want the person who finds a device to instantly have access to all of the owner's websites, emails, pictures, and so on.

Solution: Encryption

Since Chromium OS is built on Linux, we can leverage existing solutions for encrypting the user's data at the underlying operating system level. Given the available implementation choices, many possible approaches were considered (see Appendix D for

details). Of those approaches, we chose a solution that provides file system-level encryption of home directories for each user on a device.

Encrypting each user's home directory means that all of the data stored by the Chromium-based browser will be encrypted by default. In addition, data created and maintained by plugins, like Adobe Flash, as well as any data stored by HTML5-enabled web apps will be transparently encrypted without developers or users needing to take any special action.

7.2 Reclaiming lost space

Reclaiming space lost to file system fragmentation has always been a challenge with the sparse file implementations on Linux. There have been numerous discussions about supporting device calls that will tell the kernel, and then the backing device, to release unused blocks. At present, no agreed upon solution has been implemented in a way that meets our needs. The only other option in use is crawling the file system metadata manually and writing zeroes to any freed blocks. This is not only costly in I/O but also probably detrimental to SSDs.

This is another area in which ext4 has a chance to shine. ext4 explicitly supports file system defragmentation and even supports (online) defragmentation of mounted file systems. In addition, it is designed to minimize fragmentation where possible during file allocation.What does this mean for sparse files?

If a file system image is kept as defragmented as possible, the required data footprint for the image should be minimal. To that end, we will keep an event-triggered daemon running that checks for fragmentation in the background while the user is creating or deleting data. If the file system has started to become fragmented enough to benefit from defragmentation, it will perform an online defragmentation. This will keep the file system footprint minimized. On logout, the file system will be resized to this minimal size, the file truncated, and then re-extended, sparsely. An online resize will occur at next login and the wasted data will be recovered.

While this sounds nice and tidy, there are a number of implementation details that will need to be dealt with. The fragmentation monitoring and defragmentation tasks may need to occur only when the device is connected to an AC adapter. The same goes for resizing on logout. It also won't make sense to waste any time defragmenting or reclaiming lost space if a device is single user. The space in the image will always be reusable by the same user.

The last issue that will need attention is ensuring that minimal resizing is both reliable and quick. Currently, resizing downward requires a forced file system check. If it is practical and possible to avoid it or speed it up, re-sparsifying the user images will be a lightweight and effective process.

We have made a concerted effort to provide users of Chromium OS-based devices with a system that is both practically secure and easy to use. To do so, we've followed a set of four guiding principles:

- The perfect is the enemy of the good.
- Deploy defenses in depth.
- Make devices secure by default.
- Don't scapegoat our users.

In the rest of this document, we first explain these principles and discuss some expected use cases for Chromium OS devices. We then give a high-level overview of the threat model against which we will endeavor to protect our users, while still enabling them to make full use of their cloud device.

7.3 OS hardening

The lowest level of our security strategy involves a combination of OS-level protection mechanisms and exploit mitigation techniques. This combination limits our attack surface, reduces the the likelihood of successful attack, and reduces the usefulness of successful user-level exploits. These protections aid in defending against both

opportunistic and dedicated adversaries. The approach designed relies on a number of independent techniques:

- Process sandboxing
    - Mandatory access control implementation that limits resource, process, and kernel interactions
    - Control group device filtering and resource abuse constraint
    - Chrooting and process namespacing for reducing resource and cross-process attack surfaces
    - Media device interposition to reduce direct kernel interface access from Chromium browser and plugin processes
- Toolchain hardening to limit exploit reliability and success
    - NX, ASLR, stack cookies, etc
- Kernel hardening and configuration paring
- Additional file system restrictions
    - Read-only root partition
    - tmpfs-based /tmp
    - User home directories that can't have executables, privileged executables, or device nodes
- Longer term, additional system enhancements will be pursued, like driver sandboxing

Detailed discussion can be found in the System Hardening design document.

7.4 Making the browser more modular

The more modular the browser is, the easier it is for the Chromium OS to separate functionality and to sandbox different processes. Such increased modularity would also drive more efficient IPC within Chromium. We welcome input from the community here, both in terms of ideas and in code. Potential areas for future work include:

- Plugins
    - All plugins should run as independent processes. We can then apply OS-level sandboxing techniques to limit their abilities. Approved plugins could even have Mandatory Access Control (MAC) policies generated and ready for use.
- Standalone network stack
    - Chromium browsers already sandbox media and HTML parsers.
    - If the HTTP and SSL stacks were isolated in independent processes, robustness issues with HTTP parsing or other behavior could be isolated. In particular, if all SSL traffic used one process while all plaintext traffic used another, we would have some protection from unauthenticated attacks leading to full information compromise. This can even be beneficial for cookie isolation, as the HTTP-only stack should never get access to cookies marked "secure." It would even be possible to run two SSL/HTTP network stacks—one for known domains based on a large whitelist and one for unknown domains. Alternately, it could be separated based on whether a certificate exception is required to finalize the connection.
- Per-domain local storage
    - If it were possible to isolate renderer access per domain, then access to local storage services could similarly by isolated—at a process level. This would mean that a compromise of a renderer that escapes the sandbox would still not be guaranteed access to the other stored data unless the escape vector were a kernel-level exploit.

7.5 Web app security

As we enable web applications to provide richer functionality for users, we are increasing the value of web-based exploits, whether the attacker tricks the browser into giving up extra access or the user into giving up extra access. We are working on multiple fronts to design a system that allows Chromium OS devices to manage access to new APIs in a unified manner, providing the user visibility into the behavior of web

applications where appropriate and an intuitive way to manage permissions granted to different applications where necessary.

- User experience
    - The user experience should be orthogonal to the policy enforcement mechanism inside the Chromium browser and, ideally, work the same for HTML5 APIs and Google Chrome Extensions.
- HTML5/Open Web Platform APIs and Google Chrome Extensions
    - We're hopeful that we can unify the policy enforcement mechanisms/user preference storage mechanisms across all of the above.
- Plugins
    - We are developing a multi-tiered sandboxing strategy, leveraging existing Chromium browser sandboxing technology and some of the work we discuss in our System Hardening design document.
    - Long term, we will work with other browser vendors to make plugins more easily sandboxed.
    - Full-screen mode in some plugins could allow an attacker to mock out the entire user experience of a Chromium OS device. We are investigating a variety of mitigation strategies in this space.

As HTML5 features like persistent workers move through the standards process, we must ensure that we watch for functionality creeping in that can poke holes in our security model and take care to handle it appropriately.

7.6 Phishing, XSS, and other web vulnerabilities

Phishing, XSS, and other web-based exploits are no more of an issue for Chromium OS systems than they are for Chromium browsers on other platforms. The only JavaScript APIs used in web applications on Chromium OS devices will be the same HTML5 and Open Web Platform APIs that are being deployed in Chromium browsers everywhere. As the browser goes, so will we.

7.7 Secure autoupdate

Attacks against the autoupdate process are likely to be executed by a dedicated adversary who would subvert networking infrastructure to inject a fake autoupdate with malicious code inside it. That said, a well supported opportunistic adversary could attempt to subvert the update process for many users simultaneously, so we should address this possibility here. (For more on this subject, also see the File System/Autoupdate design document.)

- Signed updates are downloaded over SSL.
- Version numbers of updates can't go backwards.
- The integrity of each update is verified on subsequent boot, using our Verified Boot process, described below.

7.8 Verified boot

Verified boot provides a means of getting cryptographic assurances that the Linux kernel, non-volatile system memory, and the partition table are untampered with when the system starts up. This approach is not "trusted boot" as it does not depend on a TPM device or other specialized processor features. Instead, a chain of trust is created using custom read-only firmware that performs integrity checking on a writable firmware. The verified code in the writable firmware then verifies the next component in the boot path, and so on. This approach allows for more flexibility than traditional trusted boot systems and avoids taking ownership away from the user. The design is broken down into two stages:

- Firmware-basedverification
  (for details, see the Firmware Boot and Recovery design document)
  - Read-only firmware checks writable firmware with a permanently stored key.
  - Writable firmware then checks any other non-volatile memory as well as the bootloader and kernel.

    - If verification fails, the user can either bypass checking or boot to a safe recovery mode.
- Kernel-based verification

  (for details, see the Verified Boot design document)
    - This approach extends authenticity and integrity guarantees to files and metadata on the root file system.
    - All access to the root file system device traverses a transparent layer which ensure data block integrity.
    - Block integrity is determined using cryptographic hashes stored after the root file system on the system partition.
    - All verification is done on-the-fly to avoid delaying system startup.
    - The implementation is not tied to the firmware-based verification and may be compatible with any trusted kernel.

When combined, the two verification systems will perform as follows:

- Detects changes at boot-time
    - Files, or read-write firmware, changed by an opportunistic attacker with a bootable USB drive will be caught on reboot.
    - Changes performed by a successful runtime attack will also be detected on next reboot.
- Provides a secure recovery path so that new installs are safe from past attacks.
- Doesn't protect against
    - Dedicated attackers replacing the firmware.
    - Run-time attacks: Only code loaded from the file system is verified. Running code is not.
    - Persistent attacks run by a compromised Chromium browser: It's not possible to verify the browser's configuration as safe using this technique.

It's important to note that at no point is the system restricted to code from the Chromium project; however, if Google Chrome OS is used, additional hardware-supported integrity guarantees can be made.

7.9 Rendering pawned devices useless

We do not intend to brick devices that we believe to be hacked. If we can reliably detect this state on the client, we should just initiate an update and reboot. We could try to leverage the abuse detection and mitigation mechanisms in the Google services that people are using from their Chromium OS devices, but it seems more scalable to allow each service to continue handling these problems on its own.

7.10 Mitigating device theft

A stolen device is likely to have a higher value to a dedicated adversary than to an opportunistic adversary. An opportunistic adversary is more likely to reset the device for resale, or try to log in to use the device for himself.

The challenges here are myriad:

- We want to protect user data while also enabling users to opt-in to auto-login.
- We want to protect user data while also allowing users to share the device.
- We especially want to protect user credentials without giving up offline login, auto-login, and device sharing.
- Disk encryption can have real impact on battery life and performance speed.
- The attacker can remove the hard drive to circumvent OS-level protections.
- The attacker can boot the device from a USB device.

7.11 Data protection

Users shouldn't need to worry about the privacy of their data if they forget their device in a coffee shop or share it with their family members. The easiest way to protect the data from opportunistic attackers is to ensure that it is unreadable except when it is in use by its owner.

The Protecting Cached User Data design document provides details on data protection. Key requirements for protecting cached user data (at rest) are as follows:

- Each user has his own encrypted store.
- All user data stored by the operating system, browser, and any plugins are encrypted.
- Users cannot access each other's data on a shared device.
- The system does not protect against attacks while a user is logged in.
- The system will attempt to protect against memory extraction (cold boot) attacks when additional hardware support arrives.
- The system does not protect against root file system tampering by a dedicated attacker (verified boot helps there).

7.12 Account management

Preventing the adversary from logging in to the system closes one easy pathway to getting the machine to execute code on his behalf. That said, many want this device to be just as sharable as a Google Doc. How can we balance these questions, as well as take into account certain practical concerns? These issues are discussed at length in the User Accounts and Management design document, with some highlights below.

- What *are* the practical concerns?
    - Whose wifi settings do you use? Jane's settings on Bob's device in Bob's house don't work.
    - But using Bob's settings no matter where the device is doesn't work either.
    - And if Bob's device is set up to use his enterprise's wifi, then it's dangerous if someone steals it!
- "The owner"
    - Each device has one and only one owner.
    - User preferences are distinct from system settings.
    - System settings, like wifi, follow the owner of a device.
- Whitelisting
    - The owner can whitelist other users, who can then log in.
    - The user experience for this feature should require only a few clicks or keystrokes.

- Promiscuous mode
    - The owner can opt in to a mode where anyone with a Google account can log in.
- Incognito mode
    - Users can initiate a completely stateless session, which does not sync or cache data.
    - All system settings would be kept out of this session, including networking config.

7.13 Biometrics, smart cards, and Bluetooth

We expect to keep an eye on biometric authentication technologies as they continue to become cheaper and more reliable, but at this time we believe that the cost/reliability tradeoff is not where it needs to be for our target users. We expect these devices to be covered in our users' fingerprints, so a low-cost fingerprint scanner could actually increase the likelihood of compromise. We were able to break into one device that used facial recognition authentication software just by holding it up to the user's photo. Bluetooth adds a whole new software stack to our login/screenlocker code that could potentially be buggy, and the security of the pairing protocol has been criticized in the past. Smart cards and USB crypto tokens are an interesting technology, but we don't want our users to have to keep track of a physically distinct item just to use their device.

7.14 Login

For design details, see the Login design document.

At a high level, here is how Chromium OS devices authenticate users:

1. Try to reach Google Accounts online.
2. If the service can't be reached, check an offline cache of user->password hash mappings.
3. For every successful online login, cache a mapping between the user and a salted hash of his password.

There are a number of important convenience features around authentication that we must provide for users, and some consequences of integrating with Google Accounts we must deal with. These all have security consequences, and we discuss these (and potential mitigation) here. Additionally, we are currently working through the various tradeoffs of supporting non-Google OpenID providers as an authentication backend.

Auto-login

When a user turns on auto-login, he is asserting that he wishes this device to be trusted as though it has his credentials at all times; however, we don't want to store actual Google Account credentials on the device—doing so would expose them to offline attack unnecessarily. We also don't want to rely on an expiring cookie; auto-login would "only work sometimes," which is a poor user experience. We would like to store a revokable credential on the device, one that can be exchanged on-demand for cookies that will log the user in to all of his Google services. We're considering using an OAuth token for this purpose.

For third-party sites, we would like to provide credential generation and synced, cloud-based storage.

8 Strengths and Weaknesses

8.1 Strengths of Chrome OS

- Chrome OS will run on x86 and ARM chips. Google says it's working with several hardware manufacturers to deliver netbooks running Chrome OS in the second half of 2010.
- Along with the mobile phone market, the netbook market is growing rapidly, unlike the desktop PC market
- Chrome OS will include a variety of cloud-based services to make life easier for users, including automatic backups and software updates
- The Chrome OS may also be an attempt to capitalize on emerging markets outside the U.S.,where desktop computers and laptops remain prohibitively expensive for many people.
- Speed, simplicity, and security are the key aspects of the OS, Google says, maintaining that it will start up and get you onto the Web in a matter of seconds and will have a minimal user interface.

8.2 Weaknesses of Chrome OS

- Netbooks are usually secondary computers, bought by people who already have desktops or notebooks; in that case, a smartphone is a better, more useful choice.

- Chrome OS has another problem, which sets it apart from other netbooks: It only runs browser-based apps.

- "Browsers don't yet do everything, and there are two decades of Windows applications that have been written, performing functions that can't yet be replicated in a browser

9 Future Scopes

First, they need to open Chrome OS to light weight, third-party apps—not desktop apps, but rather apps that are purpose-built to run on low-powered devices. Also, they need to change the form factor. Chrome OS devices should not be traditional, clamshell laptops; they need to be tablets or some other form factor designed for on-the-go use. Otherwise, by the time vendors come out with Chrome OS devices late next year, the market will already have moved on.

Chrome OS developers have stated that Chrome OS will be stored only on solid state devices and not on traditional hard disks. But most people still prefer hard disks than solid state devices. So, they need to sort out this problem so that Chrome OS can be stored even on traditional Hard Disks.

10 Conclusion

- Chrome OS is in no way replacing standard operating system.
- Chrome is built for netbooks and people and people who only need a computer for internet purpose.
- Chrome will pave the way for more operating systems with the cloud computing technology.

11 References

- www.chromium.org/chromium-os
- www.wikipedia.com
- www.inforamationweeklyanalytics.com
- Digit Magazine Jan 2010 issue