Security of Google Chromebook

Katherine Fang, Deborah Hanus, Yuzhi Zheng
Massachusetts Institute of Technology
Cambridge, MA 02139
katfang, dhanus, yuzhi@mit.edu

1. Introduction

The Chrome Operating System (Chrome OS) is an operating system, which is being developed by Google, that runs on specialized hardware. The entire system, hardware through software, is called the Chromebook. The Chrome OS is based on the open source project Chromium OS. Chrome OS differs from traditional operating systems in that it is designed to work specifically with web applications. We investigated and evaluated the security principles off of which Chrome OS is based. This paper documents the result of our research.

The underlying principle of Chrome OS is that more data is moving to the web dictates a move toward cloud computing. The user data lives essentially on the web, and the Chromebook is simply a "portal" into it. Thus, if the physical laptop is lost or stolen, the user can still access their data online. However, the Chromebook also allows users to access downloaded data offline, which must be kept secure. In trying to accomplish this, Chrome OS holds to one underlying security goal: make sure no one else has access to an individual users' private information. To accomplish this, Chrome OS ensures that all downloaded data is protected and that the code running on the Chromebook is safe to use. In order to avoid problems that plague traditional operation systems such as viruses and worms, Chromebook not only ensures that the code is safe, but also incorporates an autoupdate feature to seamlessly add new patches to the system.

Overall, the goal of Chrome OS is to keep all downloaded private data on the laptop private. It does so by encrypting user data and ensuring the code on the machine is safe to use through verified boot and autoupdate. The machine itself is expendable, but even in the case of a stolen or lost laptop, Chrome OS aims to ensure attackers can not extra any data from the machine.

In this paper, we first articulate the threat model. Second, we discuss how the system works, focusing on (a) how the system verifies the code is safe to use, and (b) how the user data is encrypted and separated. Finally, we evaluate the system, compare it to other operating systems, and discuss our suggestions to make Chromebook more secure.

2. Threat Model

Chrome OS focuses on protecting against two adversaries. First, it protects against the opportunistic adversary, who deploys attacks to lure users into pitfalls that will compromise their machines or use web apps to gain unwarranted privileges to the underlying system. These opportunistic adversaries do not target a specific person or enterprise. Second, it protects against the dedicated adversary, who is willing to do not only all that the opportunistic adversary will do but also to employ devices to recover data or account credentials for malicious purposes. The dedicated adversary may even go so far as to deploy network-level attacks to attempt to subvert the Chrome OS during device login or update processes. These dedicated adversaries may target data of specific person or enterprise.

2.1. Evaluation

We feel it is reasonable to cover both the opportunistic adversary and those targeting intellectual property. The opportunistic adversary is the more common case, but as more of society moves toward cloud computing, more and more important and secret data will also make the transition. Thus, it is also important to cover the dedicated adversary as it will be impossible to backtrack and strap on the necessary security later on. It is important to lay the groundwork now.

Chrome OS, however, mentions nothing regarding phishing attacks and other online attacks that are common in todays world. While it is understandable that an operating system may be unable to prevent such attacks any more than a browser, it seems odd that Chrome OS will leave this out when it tries to make some guarantees about the Chromebook being safe to use. Chrome OS pushes the responsibility of securing the safety in the web-based attacks to the Chrome Browser and contains the effect of the attacks through autoupdating and verified boot.

2.1.1 User Analysis

The main features that Google highlights in the Chromebook marketing videos are the following: no anti-virus software or system update needed, simple to use, long battery life and portability. These features imply two main uses: a primary device for an inexperienced user or a secondary device for an experienced user. The simplicity and lack of maintenance needed for Chromebook makes it a prime candidate for people who are less familiar with the details of using computers as their primary computing tool. For example, ones grandmother. On the other hand the the portability and battery life also makes it very attractive to someone who is more versed in computers to use it as a netbook, mostly a secondary computer.

For someone unfamiliar with computers, it is extremely important to ensure the default setting are secure and also make it difficult for this type of user to put their data in risk. While someone who is using the Chromebook as a secondary computer might know some basic implications of certain security settings, newer computer users might be attracted to features that are easier to use but likely to drastically reduce security.

2.2. System overview

The Chrome browser is the only real "user application" running on the Chrome OS with which the user interacts. All other interactions with data occur through web applications in the browser. Using this methodology, Chrome OS inherits the security of the Chrome browser. That is each web app is sandboxed and privilege separated so that each process runs in its own namespace. In addition, the Chrome browser provides tab isolation and isolated app storage resources.

Chrome OS also employs mandatory access control scheme on both the application and system level. Even though the Chrome browser is the only user application running, the mandatory access control ensures that the browser will only have access to resources it needs from the user it is currently signed in as. This prevents different web apps from messing with each other, and if one breaks, it limits the effects.

One of Chrome OS's major security goals is to ensure that the system is safe to use. The approach Chrome OS uses is verified boot. On boot up, Chrome OS checks that the firmware, kernel, and system data are all valid by checking signed hashes. In addition, there are two root partitions from which Chrome OS can boot.

If one is corrupted, the other is still safe to use, and while the system is running from one, it can update the other root partition to a new version in the background.

The other major security goal Chrome OS pursues is keeping important information secret. Chrome OS does this by keeping the data in a separate partition from the root data. Each user's data is encrypted using different keys, thus users can only access his own data.

3. Verified Boot & Autoupdate

Verified boot is used to ensure that all executed code comes from the Chrome OS source tree, rather than from an opportunistic attacker or corruption. Chrome OS uses verified boot to place a guarantee on what code is running, so that if an attacker modifies any portion of the start-up code, it is detected early, preventing the attacker from controlling the computer across multiple reboots. Verification during boot is performed on-the-fly to avoid delaying system start up. It uses stored cryptographic hashes and may be compatible with any trusted kernel. Verified boot can detect changes in the writable firmware, kernel, modules, and file system, but not changes to the user's data. Through this method, Chrome OS can prevent incomplete updates, attacks, corruption, malicious users, and crashes. We discuss how verified boot works on the firmware and kernel level, and then evaluate its security implications.

3.1. Firmware

There are several components to the firmware. First, there is the read-only (R/O) firmware, which is the first code that the computer executes on start up. Second, there is the recovery firmware (which is also read-only). Third, there are two writable (R/W) portions of the firmware (A and B). Firmware A is the primary R/W firmware and is checked first. Firmware B acts as a backup and is copied to A is A is corrupted. The general flow is for R/O firmware to check the validity of the R/W firmware. If the R/W firmware can be verified, execution will pass to it. Otherwise, the recovery firmware executes.

3.1.1 Read-only Firmware

Verified boot starts with a read-only portion of firmware, which only executes the next chunk of boot code after verification. Chromium OS enforces this guarantee by storing the firmware in an EEPROM. The read-only portion of the firmware, which includes the recovery firmware path, resides in the upper portion of the EEPROM. Since the read-only portion cannot be changed, Chrome OS tries to keep it as small as possible. It contains: the root public key, RSA verification algorithms for various combinations of 1024-8192 bit key sizes and SHA-1/256/512 message digest), SHA hashing algorithm (SHA-1/256/512), a way to write and lock a portion of the NVRAM in the TPM.

The root public key is a 8192-bit RSA key and is changed by Google every 4 years. When Google pushes updates using this key, it can check the SKU of R/O firmware to see what private key it needs to sign with. This key is used as rarely as possible to keep it as secret as possible. Instead, a second 2048-bit RSA signing key (which is signed by the root key) is used to check most other information.

The reason the RSA and SHA algorithms are stored in the R/O firmware is because these algorithms do not change, and storing them in the R/W firmware means that there is a chance of modification.

The reason the R/O firmware needs to write and lock the NVRAM is to store counters. There are two major counters. First, there is the key version number (K#). The K# is a monotonically increasing number, and increments every time the signing key changes. The second is a firmware version number (F#) which comes with firmware updates that is updated every time the firmware changes. Together, K# and F# prevent firmware rollbacks as the attacker cannot replace the firmware with an older version without detection.

3.1.2 Read-Writable Firmware Verification

To verify the R/W firmware, the R/O firmware checks a header (see Figure 1). This header stores:

- *length* of header,
- *algorithm* that specifies the key size and the hashing algorithm to use,
- *signing key* that is actually used,
- *key version number*, and
- *crypto hash*, which is a cryptographic hash of the rest of the header.

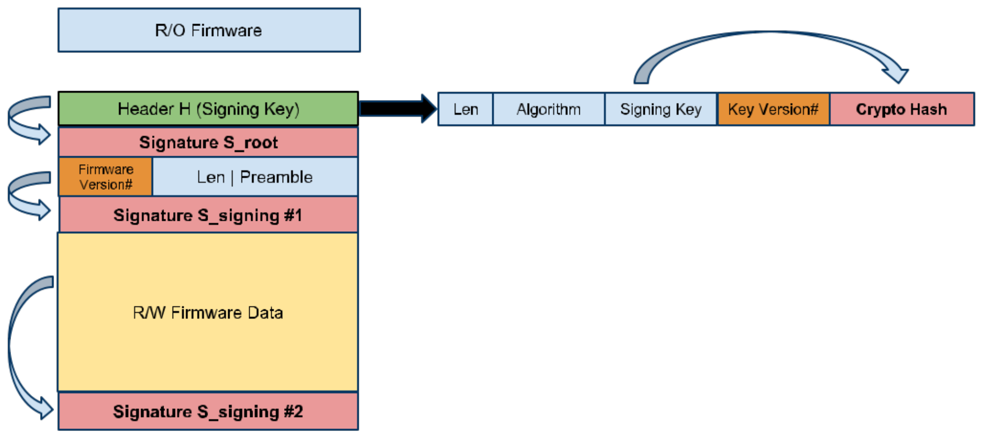


Figure 1. Overview of the data needed by R/O firmware to verify R/W firmware. The header's hash is signed by the root key, which cannot be changed after manufacture. The rest is signed by the signing key specified in the header. [**?**]

In addition, a signature of this header (signed by the root key) is stored, and the R/O firmware verifies this header with the root key. It also checks that the key version number in the header is not less than the one stored in NVRAM.

Once the header is verified, the R/O firmware can confidently use the signing key to verify other signatures. The signing key is used to sign (firmware version number, length, preamble) as well as to sign the hash of the R/W firmware. The R/O firmware checks that both of these signatures are valid, and then checks that the firmware version number is valid. Finally, it computes the hash of the of the R/W firmware and checks it against the signed version. Only when this step is verified does execution pass to R/W firmware.

3.1.3 Read-Writable Firmware

The role of R/W firmware is to verify the non-volatile memory. This includes the kernel, initrd, and the master boot record. These are verified similarly to how R/W firmware is verified. For example, a hash is computed over the kernel code and then compared against a signed hash. The signed hash is signed by a separate kernel key, which, in turn, is signed by the signing key. This key, the initrd key, and other similar keys change often to further ensure security. Because they are frequently changed, these keys are stored next to the kernel and initrd, rather than in the firmware. The alternative would be to use the signing key stored in the R/W firmware. However, this would require a firmware update each time the key changes, which is harder to perform than software updates.

3.1.4 Kernel

Once the firmware has confirmed that the kernel is authentic, the kernel takes over, verifying the rest on demand. This is done by starting a transparent block device that checks each block as it is needed. Each block is 4KB and upon access, it is checked against a cryptographic hash. Once the block is verified, it is stored in the page cache. The collection of block hashes are stored either on a stand alone partition or trailing the file system.

The hash of each section of memory is stored in a Merkle Hash tree. Each hash is a SHA–1 computed over a block. Then each bundle of blocks is hashed. And finally, all the bundles are hashed as well. This enables updates in a single block with only a $O(logn)$ overhead rather than having to recompute the a hash of the entire file system each time any file is updated. While SHA–1 is not the best hashing algorithm, these checks occur after verified boot has checked very secure hashes and signatures. In addition, it is important to verify each block quickly as it is needed on demand.

Notably, verified boot does not use the TPM; however, it is possible to integrate the TPM. For example, once the kernel starts, it could initialize the PCR registers and use those for further tracking. It would also be possible to use this method to seal the disk encryption key used to unseal the users data. If verified boot fails, then it will enter recovery mode using the read-only recovery firmware. It will require user interaction in order to complete the process.

3.2. Autoupdate for firmware and software

Autoupdate is extremely important for the security of the system. Chrome OS can effectively push out security updates and minimize the window of vulnerability rather than waiting for user interaction. The firmware and softwares update mechanisms are very similar. Once updated, verified boot can detect whether the update was successful or or not.

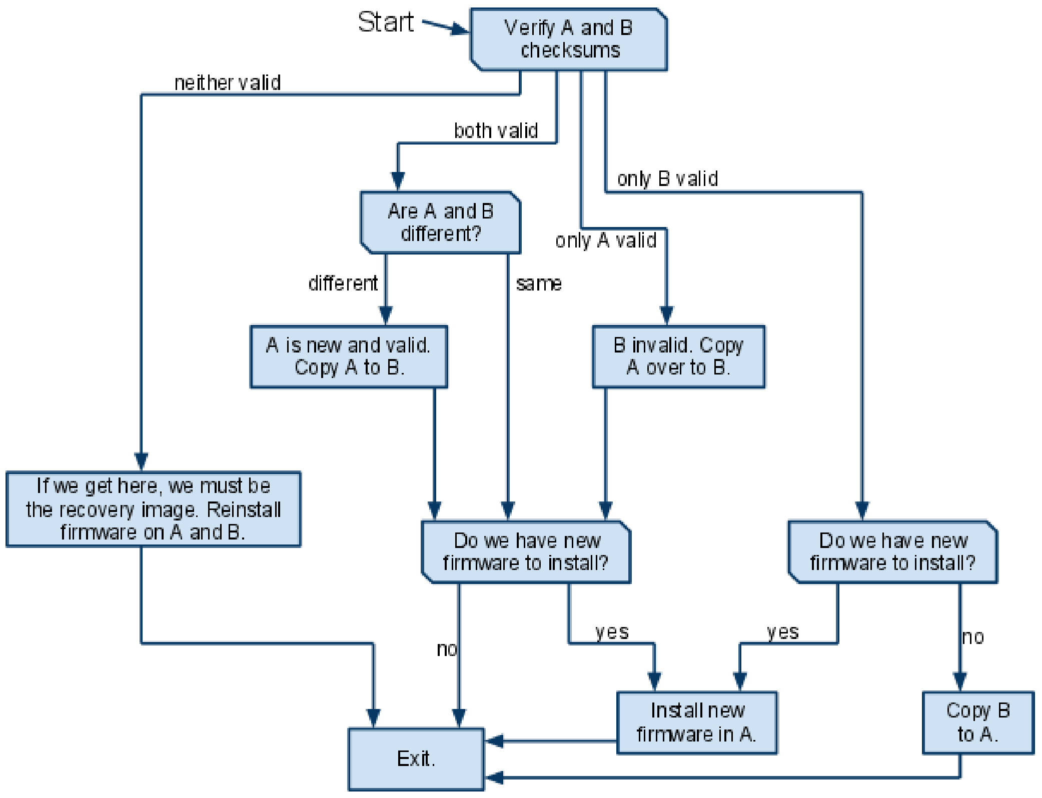
3.2.1 Firmware Update

The firmware updater is a userspace program that runs on system boot. Generally, the firmware updater will run after booting a new image that has new firmware to install. However, it is also possible to force installation of new firmware before an update.

As mentioned before, R/W firmware is divided into two parts, A and B. The system uses A primarily and uses B as a backup. That is, if A is invalid, the firmware on B will be copied over to A, and then firmware A will be used. If there is an update, A will be updated first, and if successful, the update will be copied over to B.

3.2.2 Root Update

Just as there are two copies of R/W firmware, there are are two root partitions in addition to a third partition called the “stateful partition” that stores the user data. However, instead of using one primarily and the other only as backup, the system switches between the two root partitions. The one that the user is currently booted into will be read only while the second one is called the “install” partition, with which deltas can be applied by the auto updater. This update occurs after a successful boot. However, the user is not switched to



Start
Verify A and B checksums
neither valid
both valid
only B valid
only A valid
Are A and B different?
different
same
A is new and valid. Copy A to B.
B invalid. Copy A over to B.
If we get here, we must be the recovery image. Reinstall firmware on A and B.
Do we have new firmware to install?
Do we have new firmware to install?
no
yes
yes
no
Install new firmware in A.
Copy B to A.
Exit.

Figure 2. Firmware update checks firmware A and B. Firmware A is the primary firmware and will take new updates while B acts as a backup copy.

the updated partition until after he or she reboots. When it does reboot, if the partition fails to boot a certain number of times, the system will overwrite the bad partition with the other one.

3.2.3 Security evaluation

Recall that the goal of verified boot is to ensure correctness of the currently running code. Because this correctness is only checked on boot, there are many attacks which exploit the situation in which the user does not shut down the computer after it has been attacked. In addition, only the firmware and signing key are guaranteed against rollbacks; the kernel is not. We examine a few attacks:

User leaves computer locked: If the attacker manages to attack the computer and leaves it on afterwards, then the system remains compromised until the user reboots. Chrome OS suggests that the particularly cautious user reboot often.

Replay attack: Although the firmware cannot be rolled back, it is possible to roll back the root partition. Because the hash partition is of the system image currently on the machine, if an exploit is found in some previous version and the attacker can replace the hash partition and the system image to that previous exploitable version, then verified boot will assume that the system is clean. The attacker might at this point configure the system to auto-open an attack URL to re-exploit the system immediately after reboot. The attacker has thereby thwarted the verified boot. This can be fixed a number of ways: 1) checking the version number to see if it is far too out of date before booting; 2) storing the time that the update happened using the TPMs tickstamp blob (however, auto updates cannot currently be customized per download).

Attacker switches to developer mode: The attacker can switch the machine to developer mode. It takes at least 5 minutes to launch into developer mode the first time. These 5 minutes are used both as a deterrent and to wipe the stateful partition. However, after this occurs, then the attacker can run any self-signed code

The user will not be alerted to this while the system is up and running, so if the attacker leaves the machine on, the user will not notice anything wrong. When the system reboots, the user will get a warning message that looks like the system needs to be recovered. However, this message will disappear and automatically boot after a set timeout. This attack avenue is still left open if the user 1) takes no action, or 2) does not see the reboot screen.

Overall, verified boot acts as a deterrent to persisting attacks. However, once the attacker gets the users information once, it is possible to attack them in other ways. It may perhaps be better to silently reboot every so often when the user has put it into locked mode as this will improve the chances of catching an altered system. This could be potentially difficult to time. In our experiences, booting takes 8-10 seconds, which is a reasonable amount of time to wait for a computer to come out of sleep mode. Therefore, rebooting every so often would not take too much of the users time.

4. User Data and Accounts

The main security goal of Chrome OS is to keep the users data secure. Both the data on the cloud and locally-cached data that must inaccessible to adversaries. Chrome OS keeps access to online data secure by logging in using Google Account API, and the users TPM makes offline logins possible. The local disk is secured by using encrypted files systems with separate keys and directory for each user. Those keys are secured through a few layers of encryption, utilizing both the TPM and the users password.

4.1. Login

Developers aimed to make Chromebook both secure and easy to use. For usability, it features Single-Sign-On capability to all of Googles online products. To ensure the safety of sensitive network packets, the HTTPS stack authenticates the user through Google Account API to obtain the appropriate cookies for all Google services. Once authenticated, the TPM and a hash of the password are used to wrap the magic string for future offline logins. In order to guard against network attacks, only the root certificate from Googles SSL certificates are trusted. To attack through the network, attackers will have to trick the Google registrar. The series of checks can be seen in Figure 3.

4.1.1 Offline

If the user tries to log in while offline, the system uses the TPM to check if it can successfully unwrap the magic string. If the TPM contains the wrapped magic string, the user must have had logged in on this Chromebook before. For this to work, both the wrapped magic string must be stored on the machine with the same TPM and have the correct user password. On a successful login, the user will not receive a cookie because there is no internet connection. If the user, while online, has change his password since he last logged in, the user must login using his old password in order for the magic string to be decrypted successfully. This is a permissible quirk of the system, because there is no way for Chromebook to find out about the new password without an internet connection.

Login should at least be as secure as logging in using an online browser in other operating systems. The users password is authenticated through Google each time before obtaining the cookies that will allow the user to access his data. If the Google API is somehow attacked or Googles registrar is fooled, a malicious

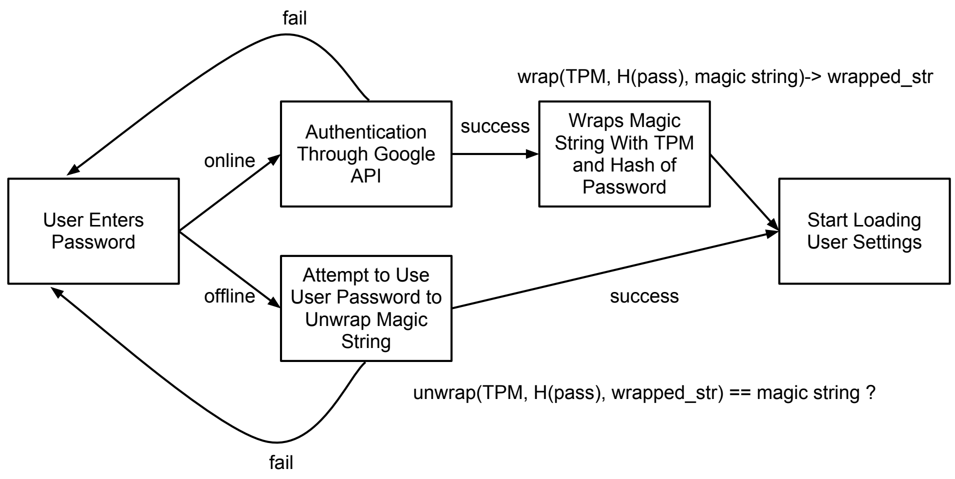

fail
wrap(TPM, H(pass), magic string)-> wrapped_str
Authentication Through Google API
success
Wraps Magic String With TPM and Hash of Password
online
User Enters Password
Start Loading User Settings
offline
Attempt to Use User Password to Unwrap Magic String
success
unwrap(TPM, H(pass), wrapped_str) == magic string ?
fail

Figure 3. The flow diagram of the checks going through the system as a user logs in

user may gain access to the users data; however, that attack would be possible in other systems as well. If that ever happens, Google will have bigger problems than just the security of the Chromebook.

4.1.2 Auto-Login

Chrome has an Auto-login option that allows the user to stay logged on constantly, even over multiple boots. Instead of caching the users password or storing long term cookies on the local drive, the system uses an OAuth token, called the übertoken, increasing security and usability. Without the übertoken, if the password is stolen, the user must change his password. In addition, there is no way to quickly invalidate long term cookies, so these cookies might remain fresh too long to reasonably keep the users data private. However, the übertoken can be revoked immediately if compromised, and it does not require additional user action like changing ones password.

While the system is in the shutdown state, the encrypted übertoken is stored with the system settings, and then the übertoken can be exchanged for cookies once the computer is back on. During login, the system notices existing übertokens and will log the user in directly. Having an übertoken demonstrates that the user has already been authenticated by Google and will log the user in automatically even while he is offline. Once logged in, the system exchanges the übertoken for a number of Google service cookies so that the user will not need to provide his password when he requests those services. This process is demonstrated in Figure 4

Although this feature increases usability, it compromises security. If the Chromebook gets misplaced or stolen, all of that users data on the cloud will be accessible to whoever holds the Chromebook. Although, theoretically, the ubertoken is revokable, the average user might not immediately understand why and how he should revoke the token. Instead, they might be more focused on the standard approach of relocating the lost machine, giving attackers a chance to steal data.

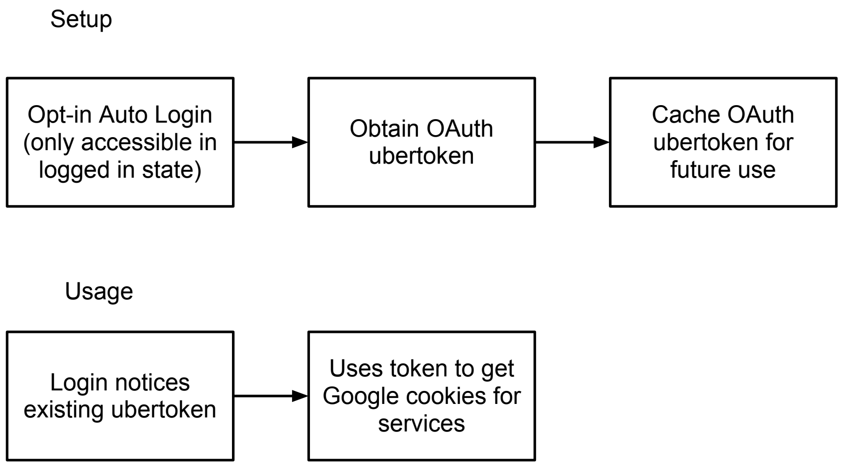

Setup
Opt-in Auto Login (only accessible in logged in state)
Obtain OAuth ubertoken
Cache OAuth ubertoken for future use
Usage
Login notices existing ubertoken
Uses token to get Google cookies for services

Figure 4. Diagram shows how auto-login is setup and retrieves cookies

4.2. User Data Encryption

The Chromebook is designed to store most user data in the cloud; however, for performance, some information, such as cached web data, local storage for web apps, and user preference settings, is kept in local persistent storage. This data must be kept securely in local storage so that even if the adversary steals a device, there is no easy way for her to read the data in plaintext. Since the data on in the cloud belongs to individual users but is not tied to any computer like traditional operation systems, there is no reason to enable the administrator access to other users data. The aim of data encryption is to ensure that only the user can access his data.

To ensure that only the user can access his data, Chrome OS claims to make the user file encryption automatic and mandatory. However, hands-on experience shows that the default is only to encrypt the password and not necessarily the synced user data. Chrome OS uses system-level encryption of home directories to keep users data safe and separate. Encryption is necessary because even in the case of an attacker, there is little they can to little to get the user data without the necessary user password. Layers of encryption is used to keep the users data safe as overviewed in Figure 5

4.2.1 Long Term User

When each user first logs into a Chromebook, he or she gets a new “vault in which to store his or her data. This “vault keeps each users data separate. The vault is stored in a directory that is a hash of user’s name and salt and is mounted using eCryptfs when the user logs in. Upon first log in, a set of keys is also created and

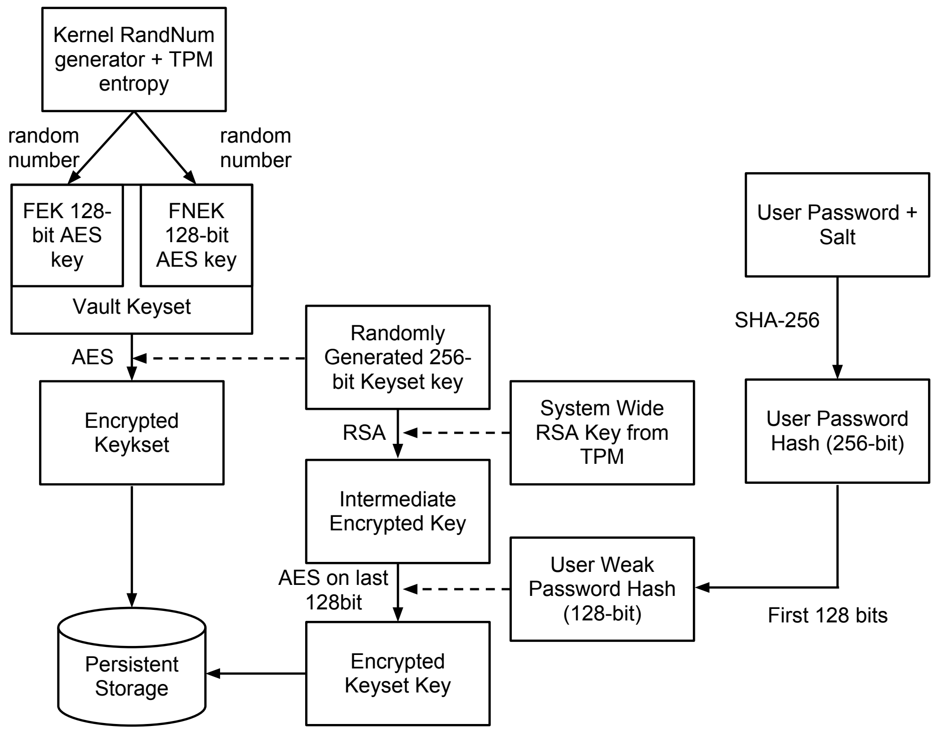


Figure 5. High level overview of the the layers of encryption for user data and encryption keys

associated with the vault for encryption and decryption functions. Two of the keys are FEK(file encryption key) and FNEK(file name encryption key). These keys are 128-bit AES keys and are created with a random generator from the kernel with some additional entropy from the TPM. These keys are stored in kernel memory for the duration of the file system mount.

The FEK and FNEK create the Vault Keyset, which must be persisted so that it is accessible across reboots. To do so securely, each users keyset is encrypted using a randomly generated 256-bit AES key before it is stored. That AES key is known as the keyset key, and it is also encrypted before it is stored. Two options can be employed to protect the user keyset key: the Scrypt method, or the TPM method. The TPM method is preferred, thus well focus on the TPM method in this paper.

The first part of the TPM method uses RSA encryption. On the first boot of the Chromebook, a system-wide RSA key is generated and wrapped with TPMs storage root key, which is then discarded. Now, only the TPM is able to unwrap the key and access the RSA private key, meaning that only this chip will be able to do these key operations and get the correct private key. The TPM is also slow, which can limit the rate of potential brute force attacks.

The second part of the TPM method uses AES encryption. The users password is salted and hashed with SHA-256. The salt is crucial for security because users passwords have limited entropy. Adding a salt increases the entropy of the hash, making it harder for attacker to brute force the key. The first 128-bitsof the SHA-256, also called the weak password hash, will be used as the AES key to encrypt the last 128-bits resulting from the RSA encryption. The resulting cypher text can be stored on persistent storage as seen in Figure 5.

To read the user data, both the user password and the original TPM is necessary to successfully decrypt the keyset key. The keyset key can then decrypt the vault keyset. The FEK and the FNEK in the valut keyset can then transparently encrypt and decrypt the users data.

Because the keysets are encrypted using the users password, if the user changes his password, the system must change the encrypted keysets. When the password changes, the old password must be entered one more time to decrypt the key before it is re-encrypted with the new password that was authenticated by Googles account API. A randomly generate AES key is more secure than using a users password hash directly because passwords have limited entropy across users.

4.2.2 Guest

For guests, any temporary browser data is stored on tmpfs and all guest browsing all is done in incognito mode. The tmpfs is never stored on disk and is erased once the guest logs out.

4.3. User Account Management

User account settings must work with the rest of the security efforts. Since each user can only see and edit his own preferences, these settings are stored in his individual storage vault, which can only be accessed with the vault keyset. When the user logs in, the users preferences are applied. Because there is no way to access user data without having the password, certain controls such as wifi preference, must be in the system settings. This makes sense if we consider that the system must be able to connect to wifi in order to get Google to authenticate the user initially before the password is first supplied.

The breakdown of system settings and user preferences is setup as below:

System Settings: locale, wifi network, owner, white-list, guest mode, proxies

User Preference: bookmarks, new tab page, browser settings, apps, extensions, themes, pinned tabs, notifications, printer, thumbnails, auto-fill data

The "owner" is the first user that logs onto the computer. An RSA key pair is generated and private key is stored in encrypted home directory of the owner. This key pair protects the system preferences, which can only be modified by the owner. All requests to change system setting needs to be signed by the private key. Furthermore, system preferences override user preferences, giving the owner more control over what is allowed on the machine.

5. Additional Modes

Until now, our discussion has been most relevant to normal mode. In addition to normal mode, through which 99% of the user interaction will happen, there are two other modes that the Chrome OS supports: recovery mode and developer mode. Having these additional modes allow the normal interactions to be more secure while enabling developers to escape the security guarantees and run their own system.

5.1. Recovery Mode

When verified boot determines that the system is in a "bad state (i.e. not developer mode, and not running code signed by Google), it will attempt to use back up copies of the firmware and software. If this does not work, it will reboot and enter "recovery mode, and run the recovery firmware. Recovery mode can also be triggered by pressing down a recovery mode button during the boot process. When this happens, the screen will show instructions on how to recover the computer as shown in Figure 6

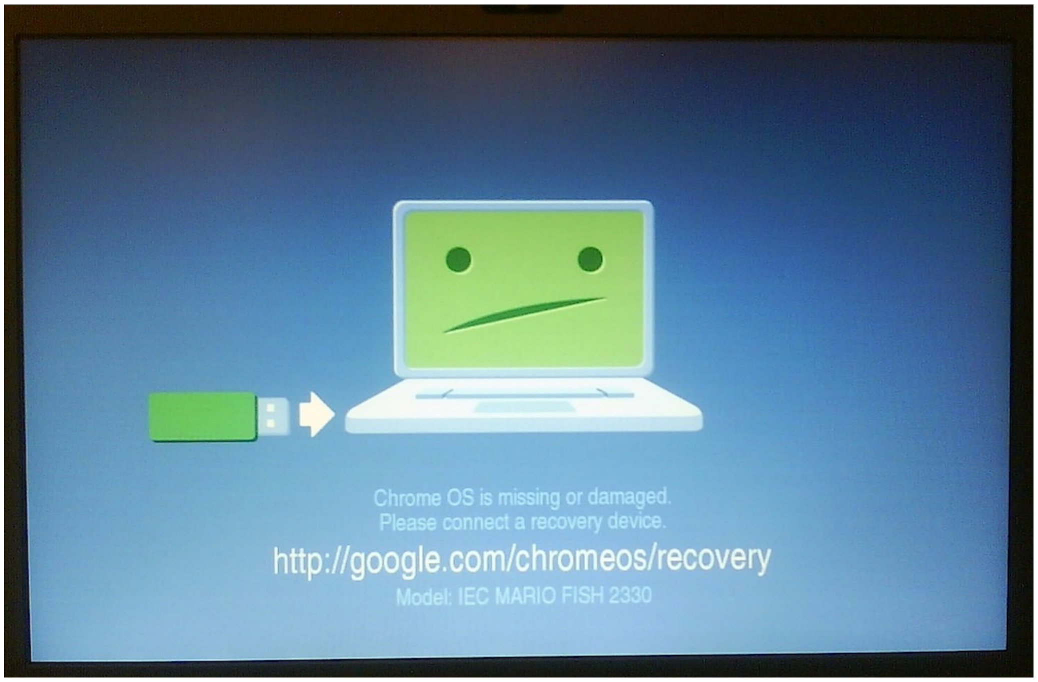

Chrome OS is missing or damaged.
Please connect a recovery device.
http://google.com/chromeos/recovery
Model: IEC MARIO FISH 2330

Figure 6. The screen shown during recovery mode

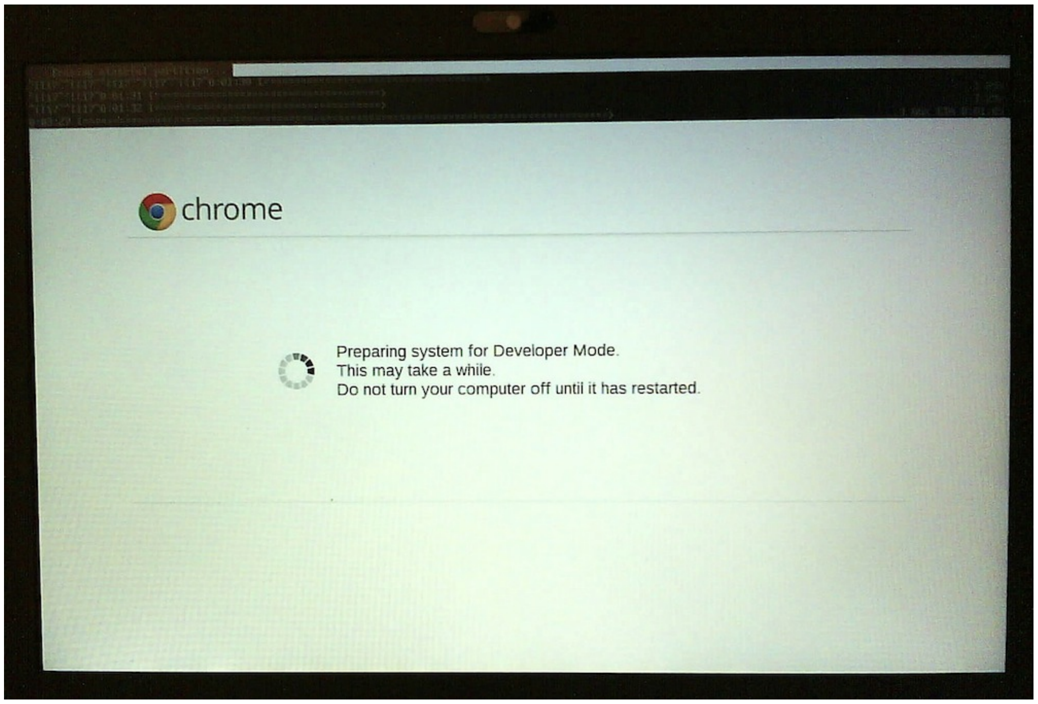
5.2. Developer Mode

By keeping developer mode separate from the normal mode, the Chromebook team allows Chrome OS to be secure for average users, but completely open for developers. Developer mode is entered by flipping a switch located behind the battery. The Chrome OS designers decided it was important for the switch to be a physical one that is not easily accessible, so adversaries must have more than a trivial amount of physical access to the machine in order to initiate an attack. No software attack would effect this physical switch.

When the developer mode switch is on, boot images that are not signed by Google can be installed. If it is the first time that the system has seen the key from the boot image, then there will be a 5 minute delay and the user data is deleted. During the wait, users will see a screen like Figure 7. If the signing key is the same as the current version, there will be no wait. The 5 minute delay is designed to make walk-by attacks more difficult to perform inconspicuously.

Chrome OS ensures that it is difficult to accidentally enter developer mode. When booting up from developer mode with code that is not signed by Google, the system shows an onerous warning message. This can only be bypassed by pressing ctrl-D or waiting 30 seconds. However, the warning message itself does not share this information with the user, instead instructions show "Press space to begin recovery, as seen in Figure 8 . If the user is inexperienced or in a hurry, he will press space without thinking. By pressing space, he will go straight to recovery mode without using unchecked code or putting the users data in danger. This way only developers who already know the shortcut will be able to quickly boot into the developer mode. The 30 second wait option is needed to allow remote developers to use the Chromebook. Hopefully, unsuspecting users will notice the 30 seconds of warning to learn that something is not quite right.

When switching back into normal mode from developer mode, the disk is wiped again (Figure 9 and



Preparing system for Developer Mode.
This may take a while.
Do not turn your computer off until it has restarted.

Figure 7. Screen during 5 minute delay on developer mode boot up

returns to using verified boot. This ensures no potentially malicious data remains on disk.

6. Comparison to Other Operating Systems

Although Chrome OS has different underlying operating principles than other operating systems, it still performs many of the same functions. Like BitLocker, Chrome OS aims to keep the data safe even if the device was stolen. One way in which Chrome OS differs from traditional operating systems is its claim that it does not need anti-virus. This this section, we give a brief comparison between Chrome OSs security features with that of other operating systems.

6.1. Chrome OS's Data Encryption vs. BitLocker's Disk Encryption

Both BitLocker and Chrome OS use the TPM and want to encrypt the user data; however, each approaches the task differently. BitLocker uses the TPM to set the PCRs and unseal the key needed to decrypt the data. It employs poor mans authentication. On the other hand, Chrome OS does *not* use the TPM to authenticate code, but instead uses the TPM to decrypt the key to the vault keyset. While BitLocker employs an optional password to encrypt the disk, Chrome OS requires the user password to decrypt each users files. BitLocker has per sector encryption,so that if one sector is decrypted, other sectors are not also easily decrypted. Chrome OS uses a similar, but different approach. Instead of ensuring each sector needs to be decrypted individually, Chrome OS has a separate vault and key for each user, meaning that decrypting one users data does not necessarily compromise other users data.

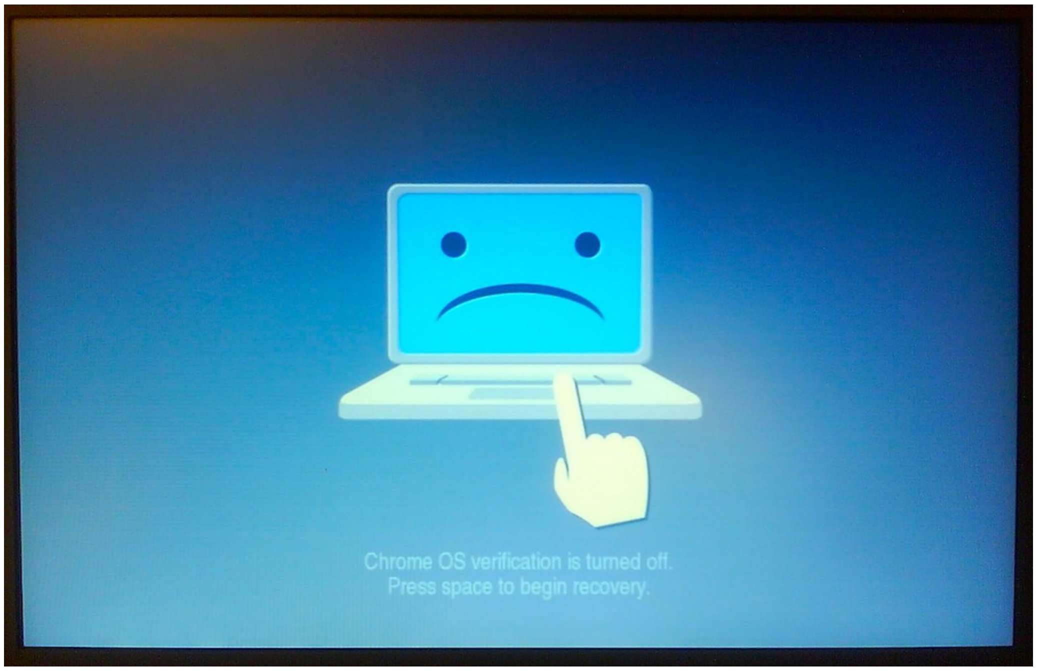

Chrome OS verification is turned off.
Press space to begin recovery.

Figure 8. The warning screen users see when the developer mode switch is on

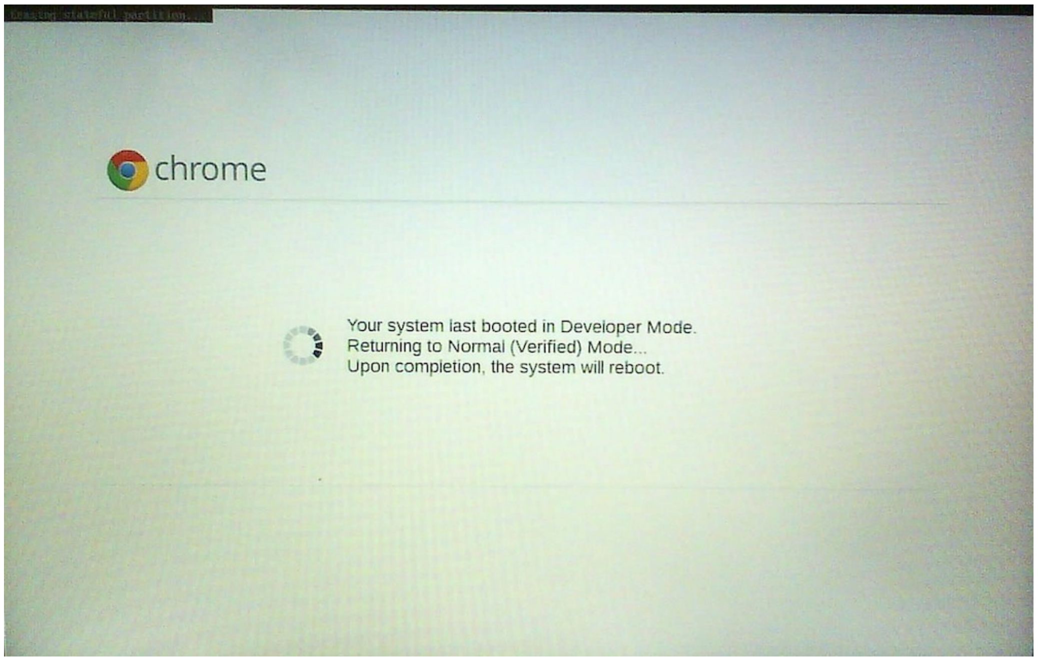
6.2. Chrome OS vs. Malware

Chrome OS claims that it does not need any anti-malware software; however, this claim is not necessarily true. Verified boot will notice if anything on the system is changed, but it may be possible to modify the browser itself though browser security vulnerabilities. It may also be possible to modify the user data. For example, an attacker might be able to secretly install a malicious plug-in without the users knowledge. It might also change the users setting to go to "attacker.com as its homepage, which redirects it to the users old homepage after performing some malicious actions. However, we believe that as time goes on, just as computers started having anti-virus software, browsers will start implementing anti-virus and anti-phishing protections.

6.3. Chrome OS vs. Windows Update

Chrome does autoupdate so beautifully that other operating systems (and programs) could learn from their implementation. There is no reason to alert the user to security patches (perhaps to feature patches). Many products have a window that pops up when you start the program saying "Would you like to upgrade your application? but this timing is wrong. The user will probably say No, because they simply want to use the program and will forget by the time they finish using the application. The next reminder occurs when they open the application, which again is bad timing for the same reason. On the other hand, Chrome OS updates in the background, ensuring that the system is kept up to date. It even has an automatic back up copy of the root partition in case something goes wrong. This keeps the window of vulnerability low. However, this update is only installed when the Chromebook is restarted.

One way they might improve their design is to notify the user than update is available, using a noticeable but unobtrusive icon, so that the security-minded user can can restart sooner than the might have otherwise,


Erasing stateful partition...

Your system last booted in Developer Mode.
Returning to Normal (Verified) Mode...
Upon completion, the system will reboot.

Figure 9. The screen that is shown when the disk is reset before restarting Chromebook in normal mode after booting it up in developer mode

but the normal user experience is still not interrupted.

6.4. Cons of Chrome OS

One of the main selling points of Chrome OS is that the important data is on the cloud, and the laptop only acts as a portal to it, which means losing the physical laptop is not a catastrophe. However, because the web is more open and connected, one may argue that is is less secure than physical computers. In recent years, the number of web-based attacks have been growing rapidly. That is to say, stealing passwords through phishing attacks, etc. is currently easier than stealing physical hard drives or breaking the cryptography. If an attacker manages to obtain a users password, the attacker can easily access all the data without even needing to have the physical computer. Given how insecure many passwords are and how easy it is to write attacks to get passwords, this may be a fundamental flaw of Chrome OS and potentially all cloud based systems. Despite this, Chrome OS does not try to address phishing or other online attacks, which means that it does not provide any stronger guarantees about the data online.

7. Chrome OS's Contribution

Chrome OS is the first cloud operation system that is shipped with commercial hardware, making new cloud-based technologies available to the general public. By connecting Chromebook with Googles other services and applications on the Chrome Browser, users have immediate access to a variety of tools when they boot their Chromebook for the first time. Chrome OSs innovative way to verify boot makes it stand out from the traditional operation systems in terms of security, ensuring users that their system is always safe to use, provided they have restarted their Chromebook recently. The autoupdate feature allows updates to be installed seamlessly, making it harder for users to evade security updates and ensuring them that their system is secure by having updates to all the recent vulnerabilities. Chromebooks focus on user privacy directly influenced their decisions in dealing with user data. To do this, each users data is encrypted separately. Unlike traditional operating systems, the owner of a Chromebook cannot access any files from other users of that Chromebook. Users credentials and data are secured by making sure no sensitive data is unencrypted.

8. Suggestions

After investigating the security of Chrome OS, we pinpointed a couple areas in which the current design could be improved. We believe that the system could be improved if it booted more often, chose better defaults, and considered some aspects of developer mode more carefully.

8.1. Boot More Often

By design, verified boot only runs on boot. However, it is possible to always avoid rebooting the computer. Given that the Chromebook only takes 8-10 seconds to boot up, it is terrible to require to reboot often, perhaps by not having the option to lock the screen. A less severe alternative would be for Chromebook to silently reboot whenever the user locks the screen or puts the machine to sleep for some amount of time (e.g. 5 minutes). In this situation, the user probably forgot to log out or is otherwise unlikely to return within the next 8-10 seconds, and a silent reboot would go unnoticed. The only downside is that the user may become annoyed if they return when the computer is in the middle of rebooting. Waiting an average of 5 seconds, however, is not much to ask for in exchange for higher security guarantees.

8.2. Better Defaults

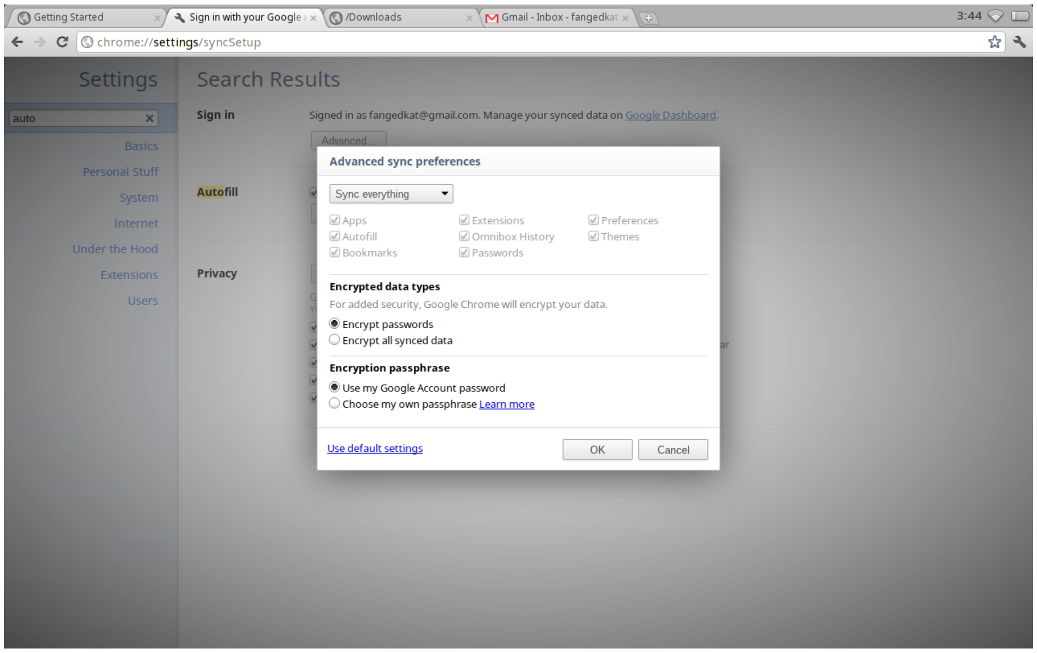
One “feature of the Chromebook is that if a user shuts the lid while logged in, they will not be logged off when the lid is opened again. It is possible to turn this option off as a user setting; however, the default is to not log the user off. This undermines the innate security of the system. The user may easily forget to log out or get in the habit of only closing the lid between uses. Instead, the default should be to require a password upon waking from sleep. Even having the system default to logging the user off after some amount of time after the user has shut the lid would be a better option. These suggestions could be combined such that if the machine is rebooted, the user will be necessarily logged out. If a user loses or misplaces his laptop with this default in place, it is less likely that an attacker will be able to access his information. The user could then decide whether or not “no password required on wake from sleep was an option he wanted to turn on.

Although Chrome OS claims to automatically enable data encryption, looking through the settings, the default encryption for data is Encrypt passwords with another option for Encrypt all synced data, suggesting that not all data is encrypted by default. Again, this undermines the innate security of the system, and Chrome OS should default to encrypt all data. The defaults can be seen in Figure 10

8.3. Dangers of Developer Mode

With access to a physical laptop, an adversary might easily implement several attacks. For example, she could boot into developer mode and leave a new, compromised OS that looked exactly the same as Chrome OS running. The user would not know unless he rebooted the computer and watched the reboot screen. Chrome OS already makes this more difficult by having a 5 minute wait time when switching to developer mode from normal mode. However, since the Chromebook is a combination of hardware and software, there could be a small LED light that normally is lit to green for “safe and red for “unsafe. The lock icon on web URLs could potentially also use this light to signal safe and unsafe sites.

The Chrome OS design documentation mentions an option for auto-login. Although we could not find a way to enable this on the Chromebook, we think this option greatly decrease the security guarantees of



Signed in as fangedkat@gmail.com. Manage your synced data on Google Dashboard.

Advanced sync preferences

Sync everything

- [✓] Apps
- [✓] Autofill
- [✓] Bookmarks
- [✓] Extensions
- [✓] Omnibox History
- [✓] Passwords
- [✓] Preferences
- [✓] Themes

Encrypted data types

For added security, Google Chrome will encrypt your data.

- [✓ Encrypt passwords]
- Encrypt all synced data

Encryption passphrase

- [✓ Use my Google Account password]
- Choose my own passphrase Learn more

Figure 10. The default data encryption setting we encountered while testing Chromebook

the Chrome OS. While we understand that auto-login may be beneficial for usability, it goes against all data encryption schemes that Chrome OS enforces. An adversary could pick up the Chromebook with auto-login enabled and steal data from it. Furthermore, inexperienced computer users may be attracted to this feature as it requires less typing to continue working, although they might not understanding the risk to their data if their computer gets stolen. Finally, inexperienced users would likely not know revoke the ubertoken if their computer were stolen.

9. Conclusion

In short, the fundamental security design of Chrome OS fundamental is solid, and it is clear the system was designed with security in mind. As data becomes more frequently digitized, security becomes increasingly important. Chrome OS rises to this challenge by making privacy guarantees about both code and data. First, verified boot ensures the code is correct, and second, a data partition with encryption to ensures that the data is safe. Autoupdate with two root partitions is a good idea not only for backup but also for security, because it minimizes the window of vulnerability.

However, Chrome OS can also improve its security without sacrificing usability. This includes actions such as restarting the computer silently while the user is not using it or setting safe defaults such as encrypting all synced data by default rather than just the password. Overall, the basic design is secure, but small implementation details render the system less secure than we would like.

10. References

http://www.youtube.com/watch?v=A9WVmNfgjtQ
http://www.chromium.org/Home/chromium-security

http://www.chromium.org/chromium-os/chromiumos-design-docs/security-overview
http://www.chromium.org/chromium-os/chromiumos-design-docs/system-hardening
http://www.chromium.org/chromium-os/chromiumos-design-docs/filesystem-autoupdate
http://www.chromium.org/chromium-os/chromiumos-design-docs/filesystem-autoupdate-supplements
http://www.chromium.org/chromium-os/chromiumos-design-docs/protecting-cached-user-data
http://www.chromium.org/chromium-os/chromiumos-design-docs/firmware-boot-and-recovery
http://www.chromium.org/chromium-os/chromiumos-design-docs/verified-boot
http://www.chromium.org/chromium-os/chromiumos-design-docs/verified-boot-crypto
http://www.chromium.org/chromium-os/chromiumos-design-docs/user-accounts-and-management
http://www.chromium.org/chromium-os/chromiumos-design-docs/login
http://seclab.stanford.edu/websec/chromium/chromium-security-architecture.pdf
http://countermeasures.trendmicro.eu/so-secure-we-dont-need-security/