ATREDIS PARTNERS

Google ChromeOS Security Competitive Analysis Report

Project Team:
Technical Testing: Jessie Chab, Tom Steele, Jordan Whitehead
Technical Editing: Shawn Moyer, Chris Bellows, Joshua Vaughn
Project Management: Joshua Vaughn

Table of Contents

Summary

Google engaged Atredis Partners to perform a competitive analysis to compare the security features of ChromeOS against other operating systems. To perform this analysis, Google supplied Atredis Partners with a document outlining security claims made by Google regarding ChromeOS. Working in conjunction with representatives of the ChromeOS team, Atredis Partners and Google selected macOS and Windows 11 as the two operating systems to use for this comparison.

To perform this analysis, flagship hardware platforms for each operating system were chosen. Where possible, Atredis Partners leveraged these devices and its operating system to show and confirm security controls and other features. For Windows 11 and macOS, this was used in conjunction with documentation review from Apple and Microsoft. For ChromeOS, open-source code repositories were used in addition to documentation.

Given the nature of this assessment, focus was placed on showing security features and controls of the platforms and performing a best-effort validation and analysis. No attempts were made to bypass or subvert these features. Past published vulnerabilities and exploitation details were used to gain an understanding of some security concepts, but they were not used to weigh the effectiveness of any feature or control.

Platform Selection

A comprehensive analysis of every hardware and software configuration would not be possible within a reasonable timeframe for this assessment. As a result, Atredis Partners selected hardware that, at the time of testing, best highlighted the security and experience of each operating system. Each device was updated, to ensure it was running the latest version of the operating system available at the time of testing. The following platforms were selected:

- Windows 11 on Microsoft Surface Laptop 5
- macOS Sonoma on Apple M1 Macbook Pro
- ChromeOS on HP Elite Dragonfly Chromebook

Key Conclusions

Atredis Partners' analysis of ChromeOS security features indicates alignment with the claims made by Google. While there were some small areas for improvement noted, they do not show a major discrepancy between the claims and the implementations of security features as tested, and instead are opportunities to further improve the security of ChromeOS.

ChromeOS is a purpose-built operating system that is designed for using cloud-hosted applications and data, and where needed, Android applications. Windows 11 and macOS are workstation operating systems, supporting a larger array of applications and workflows. Ultimately, the security of each operating system must also be measured against that flexibility, because as always, additional functionality also creates additional attack surface.

Out-of-the-box ChromeOS has a hardened default configuration and behaviors, along with a design that protects the user from unintentionally degrading system security. ChromeOS users do not need to understand the low-level security configuration or hardening options available to their device's operating system in order to have the most secure experience available. With this in mind, when contrasted with macOS and Windows 11, ChromeOS offers the most secure experience.

Nonetheless, macOS and Windows 11 can both be configured to a more secure state after installation and can match many of ChromeOS' security controls. These operating systems also feature security features that are not present in ChromeOS; however, these features exist in part out of necessity, as the threat model for each operating system is vastly different.

Claims Validation

Google provided Atredis Partners with a document with a list of security claims about ChromeOS. Each one of these claims is a statement about the current state of ChromeOS security. Atredis Partners evaluated each one with the goal of evaluating the validity of the claim. Where possible, a comparative analysis was performed against Windows 11 and macOS for similar or related functionality to those claims made by the ChromeOS team. The goal of this analysis was to draw empirical conclusions where possible and find areas where ChromeOS meets or exceeds its competition, as well as areas where ChromeOS could improve.

The following top-level categories were formulated to encompass these claims.

- User Data Encrypted at Rest
- Verified Boot
- No Privileged User
- Google Security Chip
- Browser and Android Application Sandboxing
- Enterprise and Education Lockdown
- Seamless Updates
- Remote Access

User Data Encrypted at Rest

Encryption of user data limits the impact of a stolen or otherwise intercepted device by attempting to ensure that data on storage devices is not recoverable. Encryption in operating systems can be implemented through various methods, including encrypting the entire filesystem or confining it to designated directories. A good encryption strategy is dependent on key storage; using unsound storage strategies such as software-only components or unsecured hardware could allow an attacker to recover these keys and as a result, the underlying filesystem.

> *User does not even have the option for unencrypted user data on ChromeOS, and in the case of multiple users sharing a device, each user's data is encrypted with different credentials such that one device user (or guest) cannot observe the data of another user even if they had some exploit to become the "root" user on the device.*
>
> - *ChromeOS device owners can share the device with a guest user without fear that data from the owner's profile could be leaked to the guest user, and the guest user could also have the same confidence that the owner would not have access to the guest data after logout.*
> - *On other operating systems one can often become an administrative or root user to read the data of another user. On ChromeOS even if one user could become root through some exploit, the other user's data would still be encrypted by a key not known to that logged in user.*
>
> *Google ChromeOS Claim*

Validation

As this claim covers multiple attack scenarios and outcomes, it has been broken down into individual subsections for a more granular analysis.

User Data is Encrypted by Default

User data within the realm of ChromeOS refers primarily to the information stored on disk and in memory associated with the various actions the user performs on the device, such as browser history, favorites, downloaded files, application data, images taken with the device's camera, saved passwords, auto-fill information, preferences, etc.

When a user is logged in, this information lives within the following locations on the underlying filesystem, where {UID} is a salted cryptographic hash of the user ID.

- `/home/chronos/u-{UID}`
- `/home/user/{UID}`
- `/home/.shadow/{UID}/mount/user`
- `/home/chronos/user`

- `/home/root/{UID}` for Android-specific user data.
- `/home/.shadow/{UID}/mount/root/` for Android-specific user data.

The salt is generated during system initialization using secure random. It is stored on the filesystem at `/home/.shadow/salt` with a mask of `0644`. The use of this salt can help protect against arbitrary file writes as it makes file paths unpredictable. To determine this salted hash for a given user, the `cryptohome` command can be used in Developer Mode. On the testing device there were two users configured.

```
$ sudo cryptohome --action=obfuscate_user --user=atredistest23@gmail.com
94290c58ad465508ccf9e6271350cf4518000313

$ sudo cryptohome --action=obfuscate_user --user=atredischild23@gmail.com
0c6063c62f66338869e9052a1e48693e987979a6
```

Data is mounted to the following locations when a user is actively logged in.

```
/dev/mapper/dmcrypt-94290c58-data on /home/chronos/user type ext4
(rw,nosuid,nodev,noexec,relatime,nosymfollow,seclabel,discard,commit=600,stripe=16)

/dev/mapper/dmcrypt-94290c58-data on /home/chronos/u-
94290c58ad465508ccf9e6271350cf4518000313 type ext4
(rw,nosuid,nodev,noexec,relatime,nosymfollow,seclabel,discard,commit=600,stripe=16)

/dev/mapper/dmcrypt-94290c58-data on
/home/user/94290c58ad465508ccf9e6271350cf4518000313 type ext4
(rw,nosuid,nodev,noexec,relatime,nosymfollow,seclabel,discard,commit=600,stripe=16)

/dev/mapper/dmcrypt-94290c58-data on
/home/root/94290c58ad465508ccf9e6271350cf4518000313 type ext4
(rw,nosuid,nodev,noexec,relatime,nosymfollow,seclabel,discard,commit=600,stripe=16)
```

Each of these devices is encrypted using `dm-crypt`, which offers encryption of block devices and partitions and is part of the Linux kernel. The output of `dmsetup ls --target crypt` shows these devices.

```
dmcrypt-94290c58-cache  (254, 22)
dmcrypt-94290c58-data   (254, 21)
encstateful   (254, 7)
```

Data and cache for each user account correspond to a pair of logical volumes:

- `/dev/C3PP1KC16N7BJU5N/cryptohome--94290c58--data`
- `/dev/C3PP1KC16N7BJU5N/cryptohome--94290c58--cache`

The output of `dmsetup ls –tree` shows the relationship between these devices.

```
dmcrypt-94290c58-cache (254:22)
 `-C3PP1KC16N7BJU5N-cryptohome--94290c58--cache (254:11)
    `-C3PP1KC16N7BJU5N-thinpool-tpool (254:3)
       |-C3PP1KC16N7BJU5N-thinpool_tdata (254:2)
       |  `- (259:1)
       `-C3PP1KC16N7BJU5N-thinpool_tmeta (254:1)
          `- (259:1)
dmcrypt-94290c58-data (254:21)
 `-C3PP1KC16N7BJU5N-cryptohome--94290c58--data (254:10)
    `-C3PP1KC16N7BJU5N-thinpool-tpool (254:3)
       |-C3PP1KC16N7BJU5N-thinpool_tdata (254:2)
       |  `- (259:1)
       `-C3PP1KC16N7BJU5N-thinpool_tmeta (254:1)
          `- (259:1)
```

In addition to encryption of partitions using `dm-crypt`, the `ext4` filesystem uses `fscrypt` for transparent encryption of files and directories.

Encryption is Mandatory

The second component of the claim is that the user does not have the option to store their data unencrypted on ChromeOS. By examining the preferences available to the user, including those more advanced features accessible within `chrome://settings`, particularly experimental configuration flags in `chrome://flags`, Atredis Partners found no user-accessible method of storing this data in plaintext on the device.

Encryption Keys are Unique per User

The next claim is that, in the case of multiple users sharing a device, each user's data is encrypted with different credentials. By running `dmsetup` when logged in as each user, Atredis Partners confirmed that the encryption keys are unique.

The following is output when the user associated with the hash `94290c58` is logged in.

```
$ sudo dmsetup table --target crypt --showkeys

dmcrypt-94290c58-cache: 0 434282496 crypt aes-xts-plain64
:64:logon:dmcrypt:c4fc51d920d6a8da 0 254:11 0 1 allow_discards

dmcrypt-94290c58-data: 0 434282496 crypt aes-xts-plain64
:64:logon:dmcrypt:c4fc51d920d6a8da 0 254:10 0 1 allow_discards

encstateful: 0 131989504 crypt aes-cbc-essiv:sha256
:32:logon:dmcrypt:656e63737461746566756c 0 254:6 0 1 allow_discards
```

The following is output when the user associated with the hash `0c6063c6` is logged in.

```
$ sudo dmsetup table --target crypt --showkeys

dmcrypt-0c6063c6-cache: 0 434282496 crypt aes-xts-plain64
:64:logon:dmcrypt:1ec768de8a1fa2fd 0 254:22 0 1 allow_discards

dmcrypt-0c6063c6-data: 0 434282496 crypt aes-xts-plain64
:64:logon:dmcrypt:1ec768de8a1fa2fd 0 254:17 0 1 allow_discards

encstateful: 0 131989504 crypt aes-cbc-essiv:sha256
:32:logon:dmcrypt:656e63737461746566756c 0 254:6 0 1 allow_discards
```

Note that the key data in the output is not the actual encryption key, which is stored securely in the Google Secure Microcontroller in the device.

Encryption Protects Against Root

One security claim that elevates the security of ChromeOS for multi-user environments such as Education and the Enterprise is the idea that a device user (or guest) cannot observe the data of another user, even if root access has been gained on the device. User data is encrypted using a series of intermediate keys that use something provided by the user during login, be that a PIN, password, fingerprint, etc. As a result, obtaining root to the device alone does not allow one user to decrypt and view the files of another user. Validation of this process was conducted using code and an architectural review of the `cryptohome` directory of the ChromeOS source code.

The token used to encrypt/decrypt a user's data is loaded in TPM slot 1 when they're logged in; as seen below, the other users' tokens are not loaded and therefore listed as Slot -1.

```
$ sudo /usr/sbin/cryptohome --action=pkcs11_get_user_token_info --
user=atredistest23@gmail.com
Token properties for atredistest23@gmail.com:
Label = User TPM Token 94290c58ad465508
Pin = 111111
Slot = 1

$ sudo /usr/sbin/cryptohome --action=pkcs11_get_user_token_info --
user=atredischild23@gmail.com
Token properties for atredischild23@gmail.com:
Label = User TPM Token 0c6063c62f663388
Pin = 111111
Slot = -1
```

Though other users' tokens are not loaded when those users are not logged in, with root access, the TPM and User Secret Stash (USS) of users across the device can be accessed.

```
chronos@vell-rev4 ~ $ sudo ls -l
/home/.shadow/0c6063c62f66338869e9052a1e48693e987979a6/user_secret_stash/uss.0
-rw-------. 1 root root 532 Oct 17 14:59
/home/.shadow/0c6063c62f66338869e9052a1e48693e987979a6/user_secret_stash/uss.0
chronos@vell-rev4 ~ $ sudo ls -l
/home/.shadow/94290c58ad465508ccf9e6271350cf4518000313/user_secret_stash/uss.0
-rw-------. 1 root root 532 Jul 25 10:13
/home/.shadow/94290c58ad465508ccf9e6271350cf4518000313/user_secret_stash/uss.0
```

As mentioned earlier, because these are tied to something the user knows and provides at login, this does not allow for decryption. An exploit that can persist as root across logins could potentially gain access to other users' data post login, but this scenario is not in the spirit of this claim, and as shown in other sections is not likely to occur without a vulnerability in many portions of the system.

Users and Guests are Isolated

In a guest session, Atredis Partners looked through the plethora of system diagnostic information that's available in the browser at `chrome://about`, particularly `chrome://system`, and did not identify any remnants of other users' data, aside from output from certain logical volume diagnostic tools and logs. While these logs showed the names of the user data logical volumes based on the user ID hash, there was nothing mapping those names to the user account names. If a user could read the salt value from the filesystem, in theory they could brute force the resulting hashing using a list of usernames and domains.

Guest Data is Non-Persistent

To ensure that a guest's data is not available to other users after logout, including subsequent guests, the user data generated during a guest session must be entirely non-persistent and cleared upon termination of the session.

To identify guest-related data, Atredis Partners saved a file within a guest session and searched for references to the file on disk in developer mode while the guest session was still active. The guest data was found within the following binary file.

```
/run/cryptohome/ephemeral_data/b3ce69079fa99b130a8ec9fc9961f180d2339d8a

$ file b3ce69079fa99b130a8ec9fc9961f180d2339d8a: Linux rev 1.0 ext2 filesystem
data (mounted or unclean), UUID=080fa437-51ec-4f02-8321-95679dad2f92 (extents)
(64bit) (large files)
```

With the newly identified salted hash associated with guests, Atredis Partners searched for any other evidence of the guest data on disk. While the user data directories existed within the same locations as they did for other users, these directories were empty even when the guest session was active.

Checking the output of mount for this ID as well, Atredis Partners found `/dev/loop8` mounted in numerous places on disk with this ID, but most of these locations were empty.

- `/run/daemon-store/session_manager/b3ce69079fa99b130a8ec9fc9961f180d2339d8a`
- `/run/daemon-store/smbfs/b3ce69079fa99b130a8ec9fc9961f180d2339d8a`

After exiting guest mode, this entire filesystem blob was absent from disk; a search found no other references to the name of the downloaded file.

Comparative Analysis

macOS supports Full Disk Encryption (FDE) through its FileVault feature, while Windows 11 employs a volume encryption approach primarily through its BitLocker feature. On the tested platforms, neither was required by default.

BitLocker is supported on Windows 11 Pro, Enterprise, and Education editions. On devices with a TPM, BitLocker is enabled by default when using a work, school, or personal Microsoft account.

BitLocker encrypts entire file system volumes that may be shared by multiple users. Because of this, it is possible for someone with administrative access to the system to view the files of other users. The volume is decrypted after the pre-boot process where the system and TPM ensure the system is in a known safe state. BitLocker can be disabled by administrator users. Enterprise and educational environments can enforce the requirement to use it.

Windows 11, for all intents and purposes, has disabled the use of the traditional Guest account. It can be enabled through the Microsoft Management Console but will not be able to login locally. It is recommended that this user is not used. Windows 11 now offers a “Shared PC mode” where guest users can login and use the desktop; there is also a kiosk mode where single applications can be presented and used. This requires some advanced configuration. There are settings to enable deletion policies for these users. This is not a Windows Home feature.

Apple’s Secure Enclave that is available on modern Macs supports encryption of the file system and is not something that can be disabled. It is inherent to the system design. It prevents decryption of the internal SSD without booting from the same device.

FileVault, while not mandatory, is recommended for full device protection and can be enabled during system setup. Using an Apple ID is optional for setting up the device. If a user chooses not to enable FileVault or opts not to use an Apple ID, they can still proceed with the setup process, bypassing these features.

Once FileVault is enabled, the disk will be encrypted until an allowed user supplies their password. The users that can do this are managed using System Preferences. Once the device is booted, the disk is decrypted, a user with administrator/root privileges can access any files on the device, including those of other users.

macOS supports a guest user, although this is uncommon and must be explicitly enabled. Guest accounts cannot access the files of other users without the use of privilege escalation attacks. Files created by guest users are deleted upon logout.

Conclusion

All of the platforms reviewed support encryption of data at rest at some level, and each support it to some degree by default. With macOS and Windows there are some caveats. ChromeOS has the best out-of-box support and the only design that can prevent users who have root access from directly accessing the files of another user.

Atredis Partners did not identify any areas of improvement or discrepancies between the stated claims and the gathered evidence.

Verified Boot

Verifying that a system has not been modified by an attacker during system boot ensures that no unauthorized code is present on the system. A common goal of an attacker is to establish some persistent connection to a local or remote host. This can happen at many levels, including the BIOS and bootloaders. Modifications, such as rootkits, bootkits, etc., allow attackers to maintain a persistent connection and, with enough privileges, are difficult to prevent.

> *Verified boot ensures that users feel secure when logging into a ChromeOS device by confirming that the system is unmodified at boot-up. Verified boot starts with a read-only portion of firmware which verifies that subsequent code is properly signed to ensure that all executed code comes from the Chromium OS source tree rather than from an attacker or corruption. An attacker cannot continue to "own" a machine through permanent, local changes. To that end, on boot, the firmware and other accessible regions of the system internals are verified to be in a known good state. If they aren't, then the firmware recovery process will be initiated (or the user can request permission to proceed, which would make sense in the case of a development install, for example). Each stage can verify the integrity of the next stage, all the way to measuring that the Chrome browser that we execute has not been tampered with.*
>
> *Google ChromeOS Claim*

Validation

Verified boot begins with a read-only (RO) firmware. The security of the entire verified boot strategy depends on the integrity of this firmware. Firmware is stored in flash memory. Access to the firmware is controlled using a physical line that instructs flash to mark regions as read-only (RO). This is known as a Write Protect (WP) signal.

The WP signal in legacy systems was controlled using a physical screw on the device. Because of this, it was possible for device owners (and commonly students) to open their devices and disable the WP in a fairly trivial manner. Modern systems, including those tested by Atredis Partners, use a separate chip that is controlled by the Cr50 firmware. To disable hardware WP on modern systems, users can use a debug board or a “SuzyQ” USB device to perform closed-case debugging (CCD). The system must be in developer mode to make use of this debug cable and physical access is required.

During boot, the RO firmware extends a TPM register with the current device mode. This register can be read and validated by later components to have some assurance of the given mode for compliance.

In addition to hardware, there is software WP. Software WP will not be effective unless the hardware WP has been disabled. The status of WP can be viewed within developer mode.

```
crossystem | grep wpsw
wpsw_cur  = 1   # [RO/int] Firmware write protect hardware switch current position
```

This firmware contains coreboot and depthcharge, which are open-source bootloaders. Each of these links against a verified boot (vboot) library.

Coreboot selects a firmware to load (selection is used for backup and recovery) and validates the integrity of that firmware using a root key. The root key is stored along with coreboot on the RO flash. Depthcharge is responsible for selecting, validating, and booting a Linux kernel. The Linux kernel is validated using subkeys stored in the firmware. During this process, rollback to outdated firmware is also detected and prevented.

Understating and validating this process was conducted by performing documentation and exploratory code review. While code review identified that these processes were in place as with other portions of this document, a full code review to ensure the security of their implementation was not conducted as it is outside the scope of this assessment.

The verified boot process can be reviewed on ChromeOS by navigating to `chrome://system` and viewing the verified boot section. This is generated by running the command `/usr/bin/dev_debug_vboot`. There are a few interesting sections to this output that can further help understand and validate the process of verified boot.

Included in this output is the extraction of BIOS components from flash.

```
# Extracting BIOS components...
+ flashrom -p host -r /dev/null -iGBB:GBB -iFMAP:FMAP -iVBLOCK_A:VBLOCK_A -
iVBLOCK_B:VBLOCK_B -iFW_MAIN_A:FW_MAIN_A -iFW_MAIN_B:FW_MAIN_B
flashrom 1.4.0-devel on Linux 5.15.130-20472-g682e24dd583b (x86_64)
flashrom is free software, get the source code at https://flashrom.org

Using clock_gettime for delay loops (clk_id: 1, resolution: 1ns).
coreboot table found at 0x7688d000.
–snip–
Using regions: "FW_MAIN_B":"FW_MAIN_B", "FW_MAIN_A":"FW_MAIN_A",
"VBLOCK_B":"VBLOCK_B", "VBLOCK_A":"VBLOCK_A", "FMAP":"FMAP", "GBB":"GBB".
Reading flash... done.
```

Next, keys are extracted from the GoogleBinaryBlockHeader (GBB).

```
# Pulling root and recovery keys from GBB...
+ futility gbb -g --rootkey rootkey.vbpubk --recoverykey recoverykey.vbpubk GBB
 - exported root_key to file: rootkey.vbpubk
 - exported recovery_key to file: recoverykey.vbpubk
+ futility vbutil_key --unpack rootkey.vbpubk
Public Key file:   rootkey.vbpubk
Algorithm:         8 RSA4096 SHA512
Key Version:       1
Key sha1sum:       0b187cf7f1c803ec1cc83a149e829340054e545f
+ futility vbutil_key --unpack recoverykey.vbpubk
Public Key file:   recoverykey.vbpubk
Algorithm:         8 RSA4096 SHA512
Key Version:       1
Key sha1sum:       e3c39daf95da1940936aa29eef2d101eb09905c0
```

Subsequently, they are used to validate the integrity of the firmware.

```
# Verify firmware A with root key:
+ futility vbutil_firmware --verify VBLOCK_A --signpubkey rootkey.vbpubk --fv
FW_MAIN_A --kernelkey kern_subkey_A.vbpubk
Keyblock:
  Size:                1720
  Flags:               23 (ignored)
  Data key algorithm:  7 RSA4096 SHA256
  Data key version:    1
  Data key sha1sum:    03e9e3e848d4b0fb16e2dd14389babe2667880e1
Preamble:
  Size:                  2164
  Header version:        2.1
  Firmware version:      1
  Kernel key algorithm:  8 RSA4096 SHA512
  Kernel key version:    1
  Kernel key sha1sum:    b047ada00a098f208eaba08a9d2fbf21fe6f023b
  Firmware body size:    1946496
  Preamble flags:        0
Body verification succeeded.
# OK
#   TPM=0x00010001, this=0x00010001
# Verify firmware B with root key:
+ futility vbutil_firmware --verify VBLOCK_B --signpubkey rootkey.vbpubk --fv
FW_MAIN_B --kernelkey kern_subkey_B.vbpubk
Keyblock:
  Size:                1720
  Flags:               23 (ignored)
  Data key algorithm:  7 RSA4096 SHA256
  Data key version:    1
  Data key sha1sum:    03e9e3e848d4b0fb16e2dd14389babe2667880e1
Preamble:
  Size:                  2164
  Header version:        2.1
  Firmware version:      1
  Kernel key algorithm:  8 RSA4096 SHA512
```

```
  Kernel key version:    1
  Kernel key sha1sum:    b047ada00a098f208eaba08a9d2fbf21fe6f023b
  Firmware body size:    1956864
  Preamble flags:        0
Body verification succeeded.
```

Next, the Linux kernel is copied from disk and validated using the kernel subkey. This occurs for multiple versions of the kernel, but one is shown.

```
# Examining kernels...
# copying /dev/nvme0n1p2 to kern_0...
+ dd if=/dev/nvme0n1p2 of=kern_0
65536+0 records in
65536+0 records out
33554432 bytes (34 MB, 32 MiB) copied, 0.0508992 s, 659 MB/s
# Kernel /dev/nvme0n1p2:
+ futility vbutil_keyblock --unpack kern_0
Keyblock file:        kern_0
Signature             ignored
Flags:                23  !DEV DEV !REC !MINIOS
Data key algorithm:   4 RSA2048 SHA256
Data key version:     1
Data key sha1sum:     e20ba888da25ecf13f5f313598dee08b27552f91
# OK
#   Verify /dev/nvme0n1p2 with kern_subkey_A.vbpubk:
+ futility vbutil_kernel --verify kern_0 --signpubkey kern_subkey_A.vbpubk
Keyblock:
  Signature:           valid
  Size:                0x4b8
  Flags:               23  !DEV DEV !REC !MINIOS
  Data key algorithm:  4 RSA2048 SHA256
  Data key version:    1
  Data key sha1sum:    e20ba888da25ecf13f5f313598dee08b27552f91
Preamble:
  Size:                0xfb48
  Header version:      2.2
  Kernel version:      2
  Body load address:   0x100000
  Body size:           0xc12000
  Bootloader address:  0xd0e000
  Bootloader size:     0
  Vmlinuz header address: 0xd0e000
  Vmlinuz header size:    0x4000
  Flags          :       0
Body verification succeeded.
```

The final step in the verified boot process, to verify the integrity of the system partition, covers the last statement of this claim. This portion of the file system contains all the binaries used by the operating system, including the browser (Google Chrome), system services, and system configuration files. The integrity of this ensures that attackers cannot modify the system to establish persistence on a device.

This is done using dm-verity. This is a Device Mapper target that is part of the Linux kernel. It allows a system to verify the integrity of a block device to ensure that files have not been modified. This provides a RO file system that cannot be changed by an attacker. The Linux kernel config can be viewed by a user in developer mode and shows the use of dm-verity.

```
console= loglevel=7 init=/sbin/init cros_secure drm.trace=0x106 root=/dev/dm-0
rootwait ro dm_verity.error_behavior=3 dm_verity.max_bios=-1 dm_verity.dev_wait=1
dm="1 vroot none ro 1,0 6348800 verity payload=PARTUUID=%U/PARTNROFF=1
hashtree=PARTUUID=%U/PARTNROFF=1 hashstart=6348800 alg=sha256
root_hexdigest=66d3d411681059d060db99e218d4d40c001154f7e91b507c4a8b435e964670ff
salt=b517fcd5d9a2d10b832e6ab65c70252df90b310f8c1e928d02a04389ad0dbbcf" noinitrd
vt.global_cursor_default=0 kern_guid=%U add_efi_memmap boot=local noresume noswap
i915.modeset=1 ramoops.ecc=1 tpm_tis.force=0
intel_pmc_core.warn_on_s0ix_failures=1 i915.enable_guc=3 i915.enable_dc=4
xdomain=0 swiotlb=65536 intel_iommu=on i915.enable_psr=1 usb-
storage.quirks=13fe:6500:u
```

Comparative Analysis

Macs feature a Secure Enclave subsystem to root and enforce secure boot. The Apple SoC BootROM starts the secure boot flow and is configured by a Secure Enclave signed policy, deemed LocalPolicy by Apple documentation, that describes the security mode (Full, Reduced, and Partial) and contains the signatures used to bootstrap secure boot. The LocalPolicy includes valid signatures for both normal and recovery boot modes. The LocalPolicy is paired to a selected installation, allowing multiple versions of macOS to boot with different security policies.

By default, macOS will be configured to boot in Full security mode. In Full security mode, Apple enforces a device-specific signature, linking each application installed to the physical device via the Exclusive Chip Identification (ECID). This process requires macOS to request a unique signature from Apple's signing servers for each application installed. macOS also prevents application downgrade attacks via a security version (deemed a security epoch in Apple's documentation). Apple's signing servers will not vend a signature for an application version prior to the latest, enforcing anti-rollback protection.

In Reduced security mode, the application signatures are "global" indicating that they are not tied to an ECID and do not require/utilize the Apple signing server for each installation. The applications are signed once and may be installed on any macOS device (running in Reduced mode) at any time. Reduced mode also allows the installation and loading of third-party kernel extensions.

Finally, in Partial security mode, macOS will still enforce signatures along the boot chain but will allow the boot of images signed via the device's Secure Enclave. This mode allows booting a non-Apple operating system. In this mode, the Secure Enclave will prevent access to some decryption keys,

attempting to prevent untrusted operating systems from attacking a trusted boot Apple operating system.

macOS also features System Integrity Protection (SIP), which restricts the root user from modifying sensitive portions of the system. This is aimed at preventing malicious software from modifying system behavior. Disabling SIP requires the user boot into recovery mode. Additionally, macOS features the signed system volume (SSV) that further protects the operating system from modification.

Windows 11 supports UEFI secure boot, where each step of the boot process is responsible for verifying the next step. The intention of secure boot is to validate the UEFI drivers, the Windows Loader, and other components before they are loaded. The Windows Loader then continues the pattern of verification before loading as it loads additional Windows components.

"Secured-core PCs" can provide additional protections mitigating common attacks against secure boot. A Secured-core PC can leverage Intel's Trusted Execution Technology (TXT) (or AMD equivalents) for Secure Launch. Secure Launch lets Windows provide a root of trust back to the CPU manufacturer, requiring less trust in the device firmware while still ensuring a trusted version of Windows is loaded. The hypervisor, secure kernel, and NT kernel are all validated during this start up process. Secure Launch is not enabled by default on many Secured-core PCs, including the devices tested by Atredis Partners.

Windows 11 uses a measured boot to provide attestation that can be reported to local antivirus or an enterprise attestation authority. This can allow organizations to track a known good state for their devices and isolate potentially compromised devices. The measurements can also be utilized by the system in unison with a BitLocker-encrypted startup volume. Modifications to core components of the boot sequence will result in a failure to unseal the required encryption key.

When loading into Windows, a Virtual Secure Mode (VSM) can be used to isolate key operating system components from the rest of the system. Secure boot variables are used to ensure this feature is enabled, preventing malware from simply disabling this feature.

The use of Hypervisor-Protected Code Integrity along with secure boot allows a system to ensure that no untrusted modules can run in the kernel, ensuring a trusted platform. In an enterprise organization, administrators can enforce further policies to lock in a required configuration and specify policies that further restrict what code can run on the secured system.

Conclusion

Each of the systems have mechanisms for ensuring the integrity of the boot loader and kernel. While the mechanisms vary in their implementation, they each feature robust solutions based on a hardware root of trust.

macOS and Windows 11 each have mechanisms to protect system files from being modified and only allow trusted or signed applications from running. ChromeOS exceeds these by ensuring the integrity of the system state. With the exception of a bypass, ChromeOS's architecture prevents attackers from establishing persistent access to a device across reboots.

Atredis Partners did not identify any areas of improvement or discrepancies between the stated claims and the gathered evidence.

No Privileged User

Privileged access on an operating system could allow an attacker to make changes to the system, gain access to sensitive information, establish persistence, and potentially perform lateral movement throughout an environment. Consequently, escalating privileges and targeting users with administrative access are common techniques.

> *There is no human "root" or "admin" user on ChromeOS without enabling "Developer Mode" that makes "verified boot" a little more lax, allowing your hardware to run custom (non-Google-signed) images and access to a "root" shell. ChromeOS makes it obvious when you're in developer mode by showing a special screen at boot time that prompts you to take a specific action to begin recovery (and return to verified boot).*
>
> *Google ChromeOS Claim*

Validation

Users in ChromeOS are not associated with a Linux user ID and are not operating system users in the traditional sense. As a result, they do not have a traditional privilege model. During device setup the initial user is set as the device owner, which has some additional privileges in comparison to subsequent users and guests; these include setting the time zone, controlling user access and guest sign-in settings, and wireless network preferences.

This supports the claim that there are no root or admin-level human users. Services and processes run as the `root` or `chronos` users, and the `chronos` user has `sudo` access. Access to these users is not available via the desktop UI, and non-developer-mode systems cannot switch TTYs or otherwise access a system terminal to log in as these users. Access is controlled in many places by reading the `cros_debug` variable that is stored in NVRAM.

This value can be seen by opening the Chrome Browser to `chrome://system/#bios_info` and expanding the information. All system configuration options will be displayed with their current values, including `cros_debug`.

Services use the `crossystem` command to get the status of this variable. For example, TTY availability can be observed in the following file, available in `/etc/init/console-ttyS0.conf`.

```
# Copyright 2014 The ChromiumOS Authors
# Use of this source code is governed by a BSD-style license that can be
# found in the LICENSE file.

description     "Terminal login service for VT"
author          "chromium-os-dev@chromium.org"

# The console doesn't need any of the services provided by boot-services,
# so it is safe to start it in parallel. We don't want the serial console
# to wait for boot-services to start because if boot-services fails, we
# want a serial console to debug the failure. We also don't want to use
# failsafe because that requires boot-services to have started.
start on starting boot-services
stop on stopped boot-services
respawn
oom score never

env TTY_BAUD_RATE=115200

# This file is generated at build time by chromeos-base/tty.
script
  if crossystem "cros_debug?1"; then
    exec agetty "${TTY_BAUD_RATE}" %PORT% linux
  else
    stop
    exit 0
  fi
end script
```

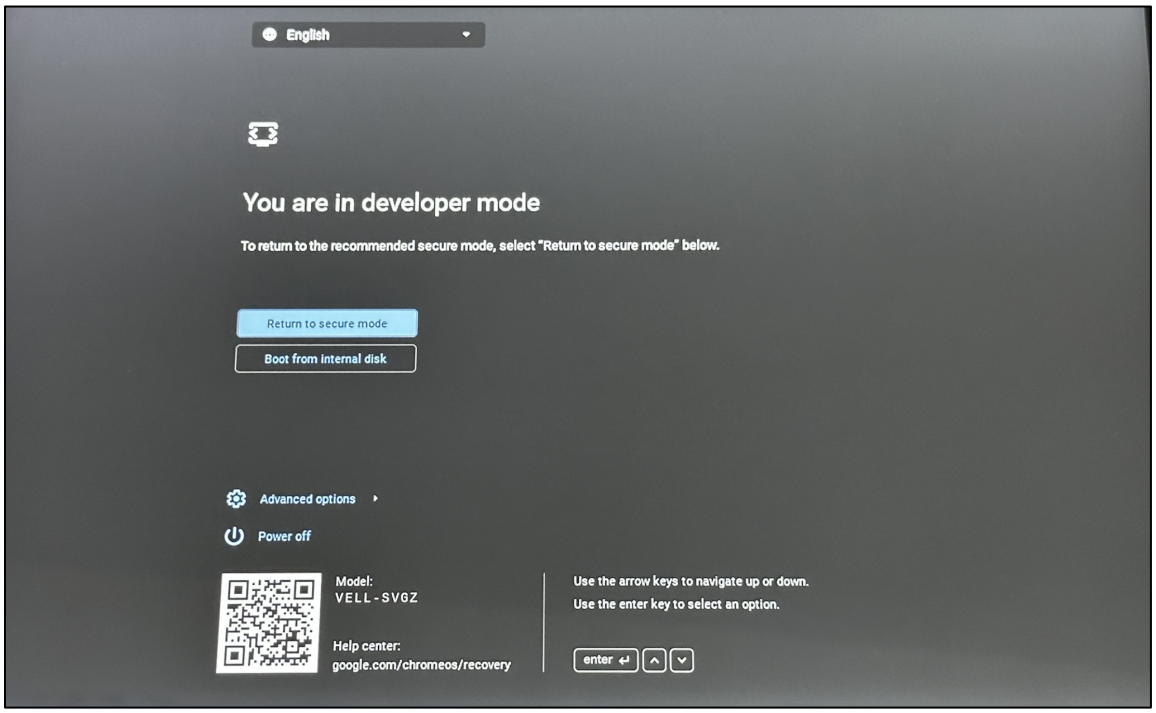
PAM configuration rules also check this value before allowing users in `/etc/shadow` (e.g., `root` and `chronos`) to login.

```
# If we're not in dev-mode, skip to the system password stack.
auth [success=ignore default=4] pam_exec.so \
  quiet seteuid \
  /usr/bin/crossystem cros_debug?1

# Check if a custom devmode password file exists and prefer it.
auth [success=ignore default=1] pam_exec.so \
  quiet seteuid \
  /usr/bin/test -f /mnt/stateful_partition/etc/devmode.passwd

# If we get to pwdfile, use it or bypass the password-less login.
auth [success=done default=2] pam_pwdfile.so \
  pwdfile /mnt/stateful_partition/etc/devmode.passwd

# See if the account exists in /etc and does not yet have a system password
# set. Only then will we allow password-less login access (see below).
# For accounts not listed in /etc, or that have a password, we do not want
# to allow them to log in.
auth [success=ignore default=1] pam_exec.so \
  quiet seteuid \
```

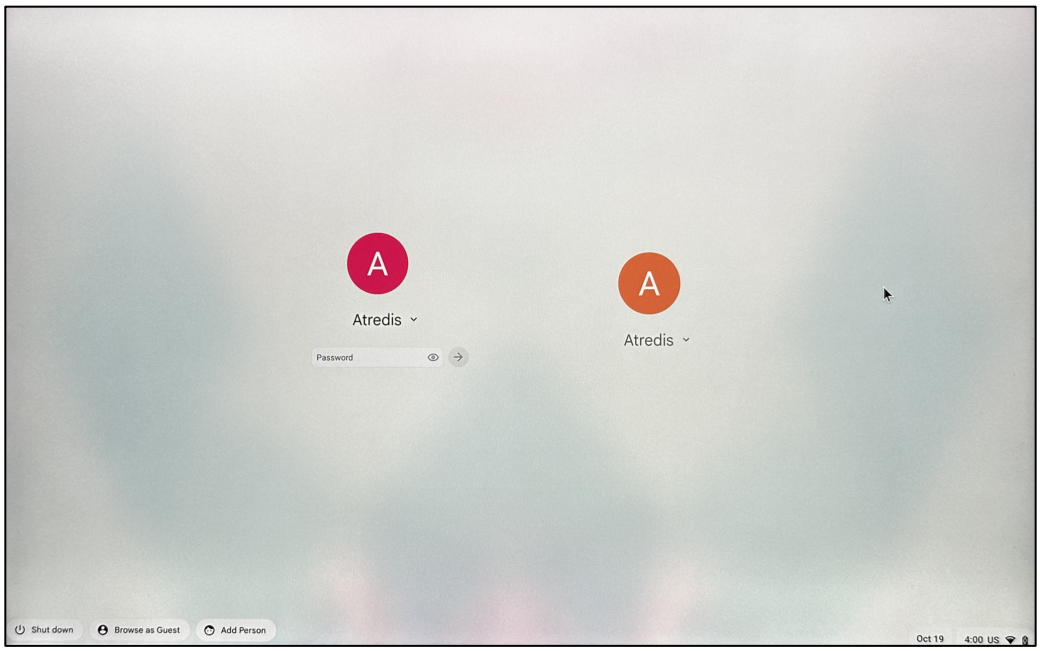
```
  /usr/bin/awk -F: [ \
    BEGIN { ret = 1 } \
    $1 == ENVIRON["PAM_USER"\] && $2 == "*" { ret = 0 } \
    END { exit ret }] /etc/shadow

# If we get here, allow password-less access
auth sufficient pam_exec.so \
  quiet seteuid \
  /usr/bin/crossystem cros_debug?1

# Fallback to a system password if one was stamped in after initial build.
```

What this ultimately means is that many classes of attack are eliminated. For example, it is not possible for an attacker to coerce a user into downloading and executing privileged code. The attack surface is not eliminated entirely; an attacker armed with a ChromeOS or Chrome exploit chain could in theory obtain privileged code execution. This greatly increases the cost and difficulty of exploitation.

When booting a device into developer mode, it is obvious to the user that they are operating in this mode. The following picture clearly shows the message to the user.



However, once the boot process is complete there is no immediate or visual indication that the device is operating in developer mode. Any indicator here cannot be trusted, as the device could already have been booted into a custom image.



Comparative Analysis

It is difficult to perform a comparative analysis here, as the paradigms are very different. Windows 11 and macOS both use a typical access control model historically used by operating systems. There are various mechanisms in place for both platforms to control access to privileged behavior, but they are not discussed as they are not relevant to ChromeOS's paradigm.

On both platforms users can be given administrative privileges. For all intents and purposes there must be at least one administrator account on the system. This account type is required to install software, make configuration changes, add accounts, etc. User accounts can be created that are not administrators and can be provided fine-grained control of the system.

There is no comparison to be made for developer mode for either the Windows 11 or macOS platforms.

Conclusion

Windows 11 and macOS support a traditional model of users and user privilege levels. The concept of a user within ChromeOS is very different in that ChromeOS users are not actually system users and subsequently can only execute applications that are available within the UI. This can heighten the security of ChromeOS users as by default dangerous permissions are not easily attainable without a costly or difficult exploit chain.

Atredis Partners did not identify any areas of improvement or discrepancies between the stated claims and the gathered evidence.

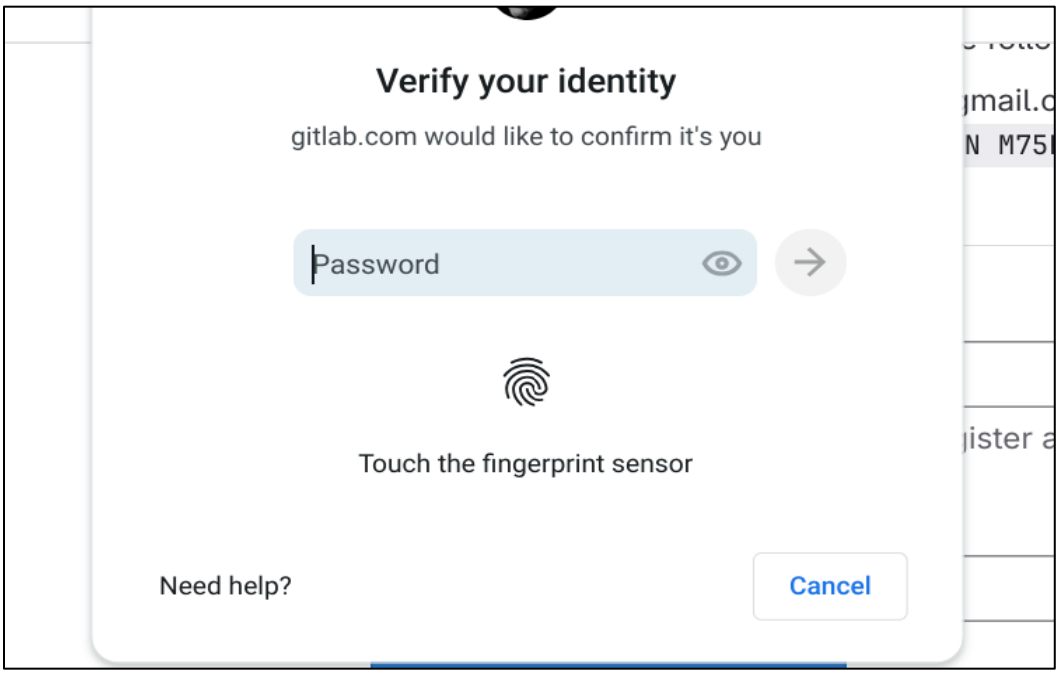
Google Security Chip

Having a dedicated security chip means an attacker must have physical access to the device to attack sensitive information secured by the security chip, and even then, it is a difficult task. On ChromeOS devices secured by the Google Security Chip, the sensitive data secured by the security chip includes two-factor authentication credentials and user data encryption keys. The security chip is also instrumental in attesting the device mode, so that a device cannot claim it is in normal mode while it is in developer mode.

> *Google security chip - available for authentication with both google and other websites for non-phishable two-factor authentication that would require a power button press in addition to other credentials. The Google Security chip protects access to user data and encryption keys. Even if an attacker had your password and hard drive, they would not be able to decrypt your data on a different device. Brute force attacks are also prevented by the Google Security Chip so that attackers can't try millions of combinations of passwords or pin codes to attempt to log in to the device.*
>
> *Google ChromeOS Claim*

Validation

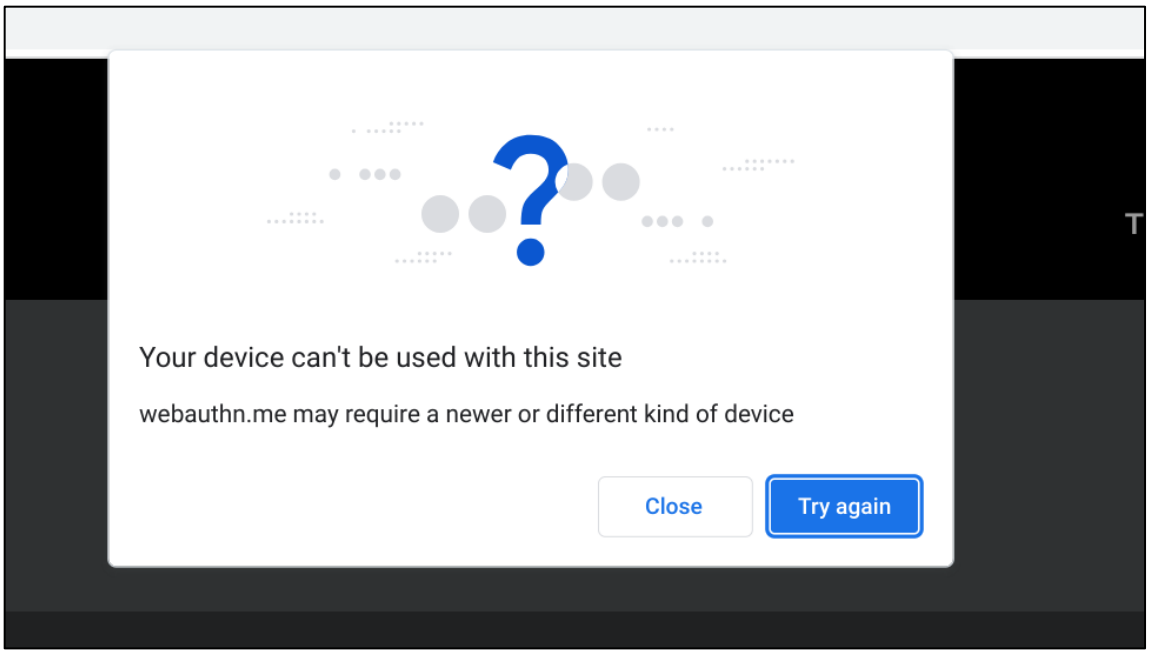
The Google Secure Microcontroller (H1) supports many security features throughout ChromeOS. Included in these is the ability to act as a Universal 2nd Factor (U2F) key. This can be used to set up authentication to third-party sites. Historically the power button was used to simulate physical presence, however, in newer systems a PIN or fingerprint may be used. Atredis Partners validated the usability of this using the Chrome browser and several sites.



In practice this works as designed; however, there are some sites that are not supported. This is because ChromeOS does not support client-side discoverable credentials. This is also known as Device-bound Passkeys. This was validated using a debugger at https://webauthn.me/debugger. To test this, the `residentKey` options were used.

When `residentKey` is set to “preferred” or “discouraged” registration is supported.

When `residentKey` is “required” the following message is shown in Chrome.



As shown, attestation is supported.

The H1 also protects user data stored on the device. This was validated using a combination of device review in developer mode and code review. Keys used for encrypting user data are stored on the TPM, and as a result, they cannot be exported. This portion of the claim was validated previously in the User Data Encrypted at Rest section.

Comparative Analysis

Windows 11 and later requires TPM 2.0, which is used by Windows Hello, boot attestation, drive encryption, and the cryptography framework. The TPM is a key part of the device security for secured-core PCs.

Windows Hello supports passkeys which are tied to the TPM and are protected by a PIN or biometric input. PINs or biometric access on the device are tied to the TPM so that the credentials cannot be used over the network. The PIN code is stored in the TPM, which is difficult for an attacker to access and requires physical device access. Windows Hello supports client-side discoverable credentials and attestation.

Drive encryption can be tied directly to the TPM and device state, so that a BitLocker encrypted volume will only be unlocked if the machine boots in a trusted state. The measurements recorded in the TPM must match the previously trusted state for the drive to be able to unlock automatically, otherwise a recovery code must be used.

macOS supports passkeys using Touch ID and FaceID when used on iOS. Private keys are generated as part of the passkey creation process by the Secure Enclave. Key material generated by the Secure Enclave is stored on the iCloud keychain and synchronized across devices. Touch ID supports client-side discoverable credentials and attestation. There are some limitations in attestation when using Safari, as the Authenticator Attestation GUID (AAGUID) is set to all zeros. This can prevent a site from validating the type of device that is used to generate and store passkeys. The iCloud keychain is a robust solution and is effective in preventing unauthorized access to users' passwords and passkeys.

Filesystem volumes that are protected by FileVault use keys that are only accessible from the Secure Enclave. Keys utilize a combination of the user's password and a hardware UID. Because of the Secure Enclave, all data is transparently encrypted using this hardware UID. As a result, it is not possible to decrypt a filesystem volume without having the disk attached to the same device.

Conclusion

Each platform has a hardware-based solution for ensuring that disks and filesystems cannot be decrypted on another device.

Windows 11 and macOS support a more seamless passkey experience. ChromeOS has limitations that may prevent the use of passkeys on some sites. The use and support of passkeys is a volatile and evolving landscape, and it is Atredis Partners' opinion that sites should support both solutions. Opinions differ as well on the overall security of "resident keys" and keys synchronized across devices. ChromeOS should attempt to evolve their solution (likely requiring a hardware evolution) to be more in line with the competitors, or state why they are not moving in this direction.

Atredis Partners did not identify any discrepancies between the stated claims and the gathered evidence.

Browser and Android Application Sandboxing

Modern users rely on their browsers to keep them safe and defend against untrusted sites. This defense is in depth: warning users about known malicious sites, as well as mitigating the effects of a vulnerable or malicious application. Strong sandboxing measures, especially around the browser, are important parts of the secure modern operating system.

> *ChromeOS protects against malicious webpages through safe browsing and site isolation, and malicious or compromised Android apps that attempt to take over the user's device or steal user data -- web pages and Android apps only have access to the resources the user allows, and everything a user chooses to run is sandboxed.*
>
> *Google ChromeOS Claim*

Validation

Comparing the security of modern browsers including Safari, Edge, and Chrome is a large and important aspect of security analysis. This analysis has been and continues to be covered extensively. Because of this, and the fact that Chrome is a cross-platform browser, this document focuses on the validation and comparative analysis of the claims made by ChromeOS documentation, and identifying areas where each browser may integrate with the platform and operating system.

ChromeOS employs several mechanisms to enforce process integrity and isolation, with the primary security goal of preventing malicious websites and applications from gaining access to protected user data or privileged operating system functionality.

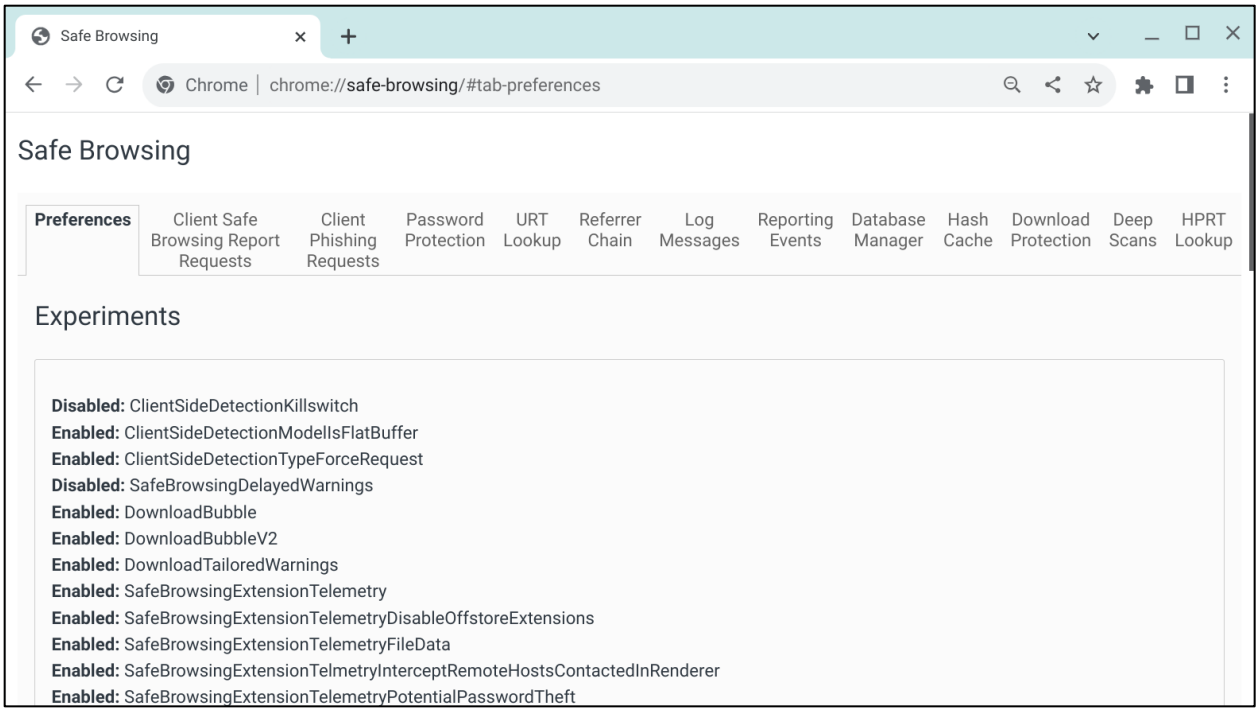
Safe Browsing and Site Isolation

Safe Browsing is a security feature of the Chrome browser that provides protection against known-dangerous websites, downloads, and extensions. This feature comes in multiple protection levels and provides some degree of customization. For example, the user can opt to have their credentials checked against compromised passwords from public data breaches in a way that does not expose this information to Google or any other third-party.

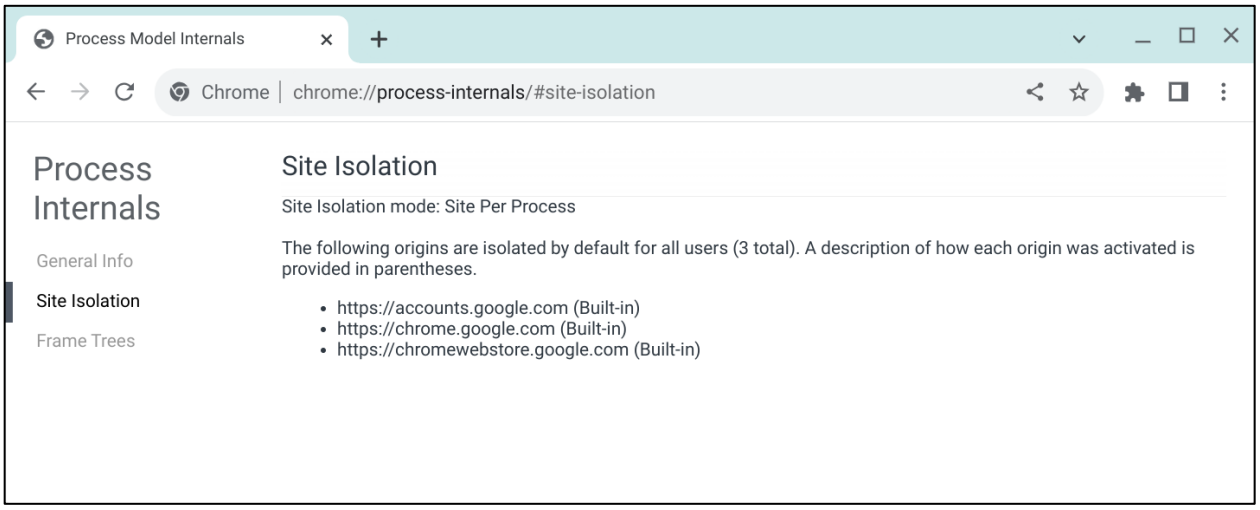
Atredis Partners found that the Safe Browsing feature was set to “Standard” by default, with the option to enable “Enhanced Protection” or disable protection entirely (including a disclaimer that revoking protection is not recommended). Google provides a decent level of transparency when presenting the different levels of Safe Browsing protection available to the user. While “Enhanced protection” is a much more proactive and thorough approach, it requires browsing data to be sent to Google for analysis. By providing the ability to opt into "Enhanced" protection rather than applying it by default,

Google is maintaining a balance between data privacy and protection, allowing users to make an informed choice about how they would like to be protected.

The `chrome://safe-browsing` page provides a plethora of real-time diagnostic and configuration information to inform users of what actions and decisions the Safe Browsing feature is performing.



In addition to warning users about unsafe sites, the Chrome browser can also take steps to mitigate damage done by a malicious site. Site isolation is a security feature in the Chrome browser that loads web pages into separate, isolated processes. On ChromeOS, the Site Isolation mode is set to "Site Per Process" by default.



With this setting, each site that a user visits will run in a dedicated rendering process. A malicious site has less opportunity to affect or gather information from other sites. Even if a site can exploit the browser to achieve code execution, they must do further work to be able to pivot from the sandboxed process and inspect the information of another site or access sensitive data. This raises the bar for meaningful attacks. For example, when loading https://www.atredis.com in a new tab, the following process was spawned.

```
u:r:cros_browser:s0            chronos  29379  0.9  1.0 1186990012 171080 ?   S<l
13:53   0:01 /opt/google/chrome/chrome --type=renderer --enable-crashpad --
crashpad-handler-pid=31708 --enable-crash-reporter=, --user-data-dir=/home/chronos
--homedir=/home/chronos/u-fa08d0fb7c79cbbe0d04aa8daaa2d2ca6f10d999 --login-
profile=user --unsafely-treat-insecure-origin-as-secure=*.linux.test --origin-
trial-disabled-features=WebGPU --change-stack-guard-on-fork=enable --enable-
logging --enable-webgl-image-chromium --log-level=1 --video-capture-use-gpu-
memory-buffer --lang=en-US --touch-selection-strategy=direction --num-raster-
threads=2 --enable-main-frame-before-activation --renderer-client-id=109 --time-
ticks-at-unix-epoch=-1701354152568391 --launch-time-ticks=16259866342 --shared-
files=v8_snapshot_data:100 --field-trial-
handle=0,i,13374301325095508805,10931272823515064104,262144 --enable-
features=ArcAdbSideloading,Borealis,CameraEffectsSupportedByHardware,Crostini,Cros
tiniGpuSupport,EnableDspHotword,EnableNeuralPalmDetectionFilter,FeatureManagement1
6Desks,FeatureManagementBorealis,FeatureManagementDriveFsBulkPinning,FeatureManage
mentFeatureAwareDeviceDemoMode,FeatureManagementGameDashboardRecordGame,FeatureMan
agementOobeSimon,FeatureManagementOrca,FeatureManagementSystemLiveCaption,FeatureM
anagementTimeOfDayScreenSaver,FeatureManagementTimeOfDayWallpaper,FeatureManagemen
tUpdateNotification,FeatureManagementVideoConference,OnDeviceSpeechRecognition,Pep
per3DImageChromium,PluginVm,QuickUnlockFingerprint,SmartDim,UmaStorageDimensions
```

Looking specifically at the contents of `/proc/29379/status`, more information can be gleaned about the various protections applied to this Chrome renderer process.

```
chronos@vell-rev4 /tmp $ cat /proc/29379/status
Name:    chrome
Umask:   0022
. . .
CapInh:  0000000000000000
CapPrm:  0000000000000000
CapEff:  0000000000000000
CapBnd:  000001fffffffffff
CapAmb:  0000000000000000
NoNewPrivs:   1
Seccomp: 2
Seccomp_filters:   1
. . .
```

When a process runs as a non-root user, as in Chrome's case, granular root-level functionality can be granted via Linux capabilities. The capabilities granted to this process are designated above by items with the Cap prefix; this process has no inherited, permitted, effective, or ambient capabilities.

Seccomp, specifically `seccomp-bpf` in the current iteration of ChromeOS, is a security feature within the kernel that can restrict the syscalls that an application can make. The Seccomp value of 2 in the status output demonstrates that Seccomp is active for this process and a policy is applied. ChromeOS uses `seccomp-bpf` to protect the kernel from any malicious code that may execute in userland.

Linux namespace information can be obtained for this process via the `lsns` command. In the following output, the renderer process PID is denoted as being part of a single namespace, particularly a PID namespace, with no other members (`NPROCS` notes the total number of processes within the namespace with the `NS` ID listed in the first column). A single process running in its own PID namespace is effectively isolated from all other processes on the system.

```
        NS TYPE   NPROCS   PID USER    COMMAND
4026534535 pid         1 29379 chronos /opt/google/chrome/chrome
```

In addition to running pages in separate renderer processes, Chrome as a whole runs in a segmented manner, where different components run in individual processes rather than a single overarching process with all requisite permissions, access, and functionality. This provides a mechanism for the browser to follow the principle of least privilege where possible. To demonstrate this, Atredis Partners observed all running processes for Chrome and identified the following individual component processes.

```
/opt/google/chrome/chrome
/opt/google/chrome/chrome --type=zygote (3)
/opt/google/chrome/chrome --type=gpu-process (1)
/opt/google/chrome/chrome --type=utility (6)
  --utility-sub-type=network.mojom.NetworkService
  --utility-sub-type=storage.mojom.StorageService
  --utility-sub-type=ash.ime.mojom.ImeService
  --utility-sub-type=ash.ime.mojom.ImeService
  --utility-sub-type=ash.local_search_service.mojom.LocalSearchService
  --utility-sub-type=quick_answers.mojom.SpellCheckService
/opt/google/chrome/chrome --type=broker (1)
/opt/google/chrome/chrome --type=renderer (5)
```

Numerous system services run under `minijail` with configuration to run under their own distinct Linux user UIDs and groups, so Discretionary Access Controls (DAC) can limit what resources these applications can process. Where necessary, root-like functionality is granted via Linux capabilities, also applied via `minijail`. `Minijail` provides several command line flags that allow Linux namespaces to be applied to these services. Namespaces allow processes to have their own isolated view of system resources, where each namespace provides a separate instance of resources like the network, filesystem, process IDs, and more, essentially creating a sandboxed environment. This namespacing provides another wall against an infected process accessing sensitive resources.

To address some of the limitations of the other mechanisms and provide a robust and granular method of Mandatory Access Control (MAC), SELinux is also leveraged in cases where stricter protections are required. When deployed correctly, in addition to process isolation, SELinux helps enforce process integrity by placing mandatory restrictions on what subjects (services and applications acting on behalf of users) have permissions to interact with and modify a process's code, data, IPC components, etc.

Atredis Partners found that although the ChromeOS system is in SELinux enforcing mode, where SELinux is blocking all actions that are not explicitly allowed by the policy, there are numerous processes running in SELinux domains that are configured as permissive, where any violations to the policy would be permitted and audited, rendering SELinux completely ineffectual for these domains, aside from auditing what actions would have been blocked if the domains were in the enforcing state.

For example, the currently deployed SELinux policy for the domain in which the Chrome browser runs contains the following line.

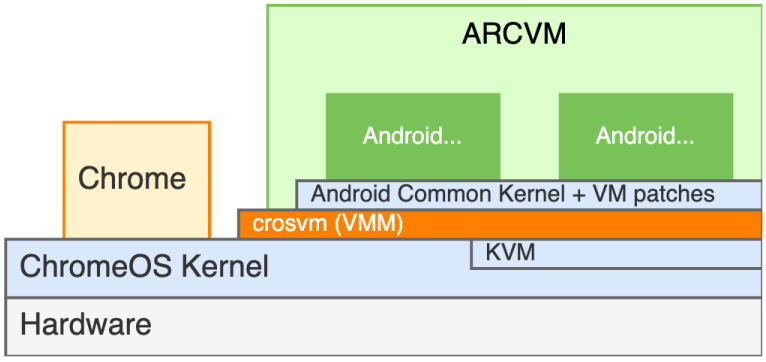
```
permissive cros_browser;
```

This directive renders all policy rules defined in the file as functionally unnecessary, where any attempted violations to the applied policy are permitted and audited. Essentially, when considering the Chrome browser, SELinux is not an active security mechanism.

That being said, the combination of Safe Browsing, Site Isolation, Linux namespaces, Seccomp policies, and Discretionary Access Control provides sufficient protection when considering the primary security goal of ensuring malicious websites cannot compromise the integrity or confidentiality of user data and core ChromeOS components.

Android Applications and the ARCVM

To support the installation of user applications while maintaining the integrity of the system, ChromeOS deploys ARCVM, an architecture that is integrated with the host in a way that allows for Android applications to run visually integrated with the desktop environment while functionally isolated from the core ChromeOS system.



Because ARCVM essentially runs an Android system, Android's built-in application and data-level isolation features apply, such as each application running as its own user, with associated data files appropriately owned by the user/group; the Android subsystem also runs with SELinux in enforcing mode, with no domains set to permissive.

The permissions model of Android also applies here, where applications have no access by default and will request access to individual components as needed, such as the microphone, camera, and storage, allowing the user to decide whether to grant access or keep it blocked. Atredis Partners confirmed that in this environment, granting the “Storage” permission specifically applies to the user’s Downloads directory and no other areas of storage. Attempts to leverage symbolic links to reach other areas of the filesystem from this directory were ultimately unsuccessful, as the user data partitions are mounted with the `nosymfollow` option.

The ChromeOS version under test was using an emulated Android 11 environment; by enabling the Linux development environment and ADB debugging, this emulated environment could be accessed like a typical unrooted Android device for further analysis. Atredis Partners leveraged this environment to further explore the Android system for any potential attack surface or weaknesses that a malicious or compromised application could exploit.

The Android data partition for each user's environment is mounted to `/home/root/{UID}/android-data/` on the root ChromeOS file system, where `{UID}` is the salted cryptographic hash of the user ID discussed previously in the User Data Encrypted at Rest section. That being said, this information is only accessible to a user when Developer Mode is enabled. Like other user data on disk, it was found to be encrypted using `dm-crypt` and only available for the currently logged-in user.

```
/dev/mapper/dmcrypt-fa08d0fb-data on
/home/root/fa08d0fb7c79cbbe0d04aa8daaa2d2ca6f10d999 type ext4
(rw,nosuid,nodev,noexec,relatime,nosymfollow,seclabel,discard,commit=600,stripe=16
)
```

Changes to the Android environment, such as installing new applications and updating app preferences, were immediately reflected on this mounted partition on disk. Atredis Partners was unable to leverage this mounted partition to otherwise access or influence unintended areas of the system via the emulated Android environment, once again largely due to the use of the `nosymfollow` mount option.

Comparative Analysis

Across all the compared operating systems, sandboxing and isolation mechanisms are used to provide protection. ChromeOS is more intimately integrated with the browser, so security paradigms are slightly different. For example, in ChromeOS there is no separate application for configuring the settings of the OS and user accounts; this is all cloud-based and performed via the browser. Rather than perform a full security comparison of the browsers available by default on Windows 11 and macOS, which has been done previously and can be found publicly, Atredis Partners focused on comparing the general application sandboxing mechanisms available on these platforms.

On macOS, in addition to standard UNIX-style Discretionary Access Controls based on user and group identifiers, System Integrity Protection (SIP) is deployed. SIP is a form of Mandatory Access Control (MAC) built into the kernel to prevent the modification of critical system files by malicious applications or users. SIP also prevents the loading of unsigned kernel extensions and prevents macOS system processes from being debugged and inspected with tools like `dtrace`. SIP can only be disabled by restarting into recovery mode and running `csrutil disable`.

macOS also implements application sandboxing that is integrated with the XNU kernel via the `Sandbox.kext` kernel extension. As explained with ChromeOS, sandboxing reduces an application's reach, hindering available malicious actions for a compromised process. Applications are required to run within a sandbox to be accepted for distribution in the App Store. Applications are not sandboxed by default on macOS, they must be configured to use a sandbox via the `com.apple.security.app-sandbox` entitlement. Sandboxed applications are limited to the actions specified within their respective sandbox profiles. There are some actions that a sandboxed application is not permitted to perform regardless of what's configured in the profile. These include privileged actions like loading kernel extensions, configuring network settings, etc. Each profile can either be in allowlist mode, where all access is denied unless explicitly granted, or denylist mode, where all access is allowed unless explicitly denied.

macOS offers Gatekeeper that by default prevents applications that are not installed from the App Store from executing. Privacy and accessibility settings prompt the users when new applications are attempting to read portions of the disk or perform various actions with privacy or security implications.

On Windows 11, along with Discretionary Access Controls based on user and group permissions, Microsoft incorporates Mandatory Integrity Control (MIC), a feature designed to regulate access to objects within the operating system. This method enforces access restrictions based on integrity levels assigned to objects throughout the system, represented by integrity SIDs (secure identifiers). Browsers on Windows will often sandbox isolated processes with restricted tokens at an untrusted integrity level. The integrity SIDs for securable objects are stored within the system access control list (SACL). Any object without an integrity SID is treated as if it had medium integrity. The integrity SIDs for security principals are stored in their access tokens.

Similar to Seccomp in the Linux kernel, Windows provides the ability to restrict what system calls a process can invoke. This is done by setting the `ProcessSystemCallDisablePolicy` via the `SetProcessMitigationPolicy` function. `ProcessSystemCallDisablePolicy` leverages the `PROCESS_MITIGATION_SYSTEM_CALL_DISABLE_POLICY` structure, which defines several flags for restricting syscall functionality. Browsers on Windows will also usually disable entire groups of system calls by disabling win32k calls for isolated processes.

In addition to the sandboxing methods, Windows also supports a security mechanism useful for preventing unwanted or unsafe applications from running. Smart App Control is a feature available on Windows 11 that checks applications against a Microsoft security service to determine if it is safe to run. If an application cannot be determined to be safe or unsafe, it will run if it has a valid digital signature, otherwise it will be blocked. By default, this service is run in Evaluation mode. This mode attempts to evaluate based on user behavior if it would be useful and chooses to set it to On or Off. Users have manual control and can set this mode in the settings menu.

Advanced users can also specify AppLocker and Windows Defender Application Control policies to further specify what applications may run on a system, and what should be denied. These policies can enforce code signing by trusted parties and effectively restrict execution on a Windows system. AppLocker can apply restrictions to not only binary executables, but libraries and other executable scripts as well. A system with proper restrictions can prevent the execution of many types of vulnerable or malicious code, which can reduce the attack surface available to an attacker seeking a foothold on a system.

Windows also provides a reputation-based system for checking applications. Reputation-based-protection is a number of security focused features that help defend against malicious software. Each of the following are optional and are on by default.

- Check apps and files: Uses Microsoft Defender SmartScreen to evaluate the repudiation of applications and files that are downloaded using a browser.
- SmartScreen for Microsoft Edge: Monitors sites and downloads a user navigates to and checks them against known or suspicious malicious sources.
- Phishing protection: Attempts to prevent users from disclosing their password into browsers and applications and protect against password reuse.
- Potentially unwanted app blocking: Unlikely other file and application-based setting, this doesn't check for malware but instead checks for software that may simply be unwanted, such as adware.
- Smart Screen for Microsoft Store apps: Uses SmartScreen on applications downloaded from the Microsoft Store.

Isolated browsing uses Microsoft Defender Application Guard to execute Microsoft Edge in a separated container using the system hypervisor. This hypervisor-based security strongly isolates the browser from the rest of the system, increasing the protections for the rest of the system should a browser process become compromised. This feature is not installed by default and is managed using the Windows Features control panel. It is not available on Windows 11 Home. Once enabled, users can launch Microsoft Edge tabs and windows in an isolated environment. Settings are available for controlling how Edge will save data, handle copy and paste, print files, camera and microphone settings, and graphics settings. Some diagnostic details are available at `edge://application-guard-internals/`. Among these is the command line used to launch Edge.

```
"C:\119.0.2151.97\edge\Application\msedge.exe" --user-data-
dir="c:\119.0.2151.97\edge\\User Data" --wdag-container --wdag-host-client-
id=55c843f3-4706-448d-b648-dcac56501e0d --wdag-host-sample-id=34583603 --wdag-
host-session-id=9 --wdag-host-ukm-client-id=0 --wdag-host-ukm-session-id=0 --wdag-
sqm-id=s:325E57CA-FA29-471F-AA7E-605CA976A265 --wdag-http-header-enabled --no-
first-run --no-default-browser-check --disable-prerender-ntp --disable-
features=WebRtcHideLocalIpsWithMdns,msRobin,msDataProtection,msEndpointDlp,msIrm,m
sIrmv2,msMdatpWebSiteDlp --silent-launch --flag-switches-begin --flag-switches-end
--wdag-navigation-id=00000000-0000-0000-0000-000002e57a27 --
```

Windows can use a hypervisor (Hyper-V) for isolation of other important services. Core security services such as the Local Security Authority (LSA) can use a hypervisor to isolate credential

information. Hypervisor-protected Code Integrity (HVCI) is another virtualization supported security tool that seeks to ensure the Windows Kernel does not load any untrusted code.

A hypervisor tool usable directly by users is the Windows Sandbox. This is a Windows feature that can be enabled by Windows 11 Pro and Enterprise users which uses Hyper-V to run a virtualized instance of Windows 11 as an isolated desktop environment. Users can run programs that are isolated from the host operating system. The sandbox is not persistent and when it is closed all data is deleted.

Conclusion

Comparing Windows 11 and macOS to ChromeOS in this context identifies key differences in the flexibility of each platform. For example, on Windows 11 and macOS it is possible to install and run third-party applications, where this is only possible in ChromeOS by using an isolated VM. Because of this, Windows 11 and macOS have a number of security features aimed at preventing the execution of malicious or unauthorized software, whereas ChromeOS takes the wholesale approach of isolating applications from the core operating system entirely.

The attack surface and protections implemented for browsers on each platform are similar and will continue to evolve. The default sandboxing of third-party applications using a VM reduces the ChromeOS attack surface in comparison to Windows 11 and macOS.

Atredis Partners did not identify any areas of improvement or discrepancies between the stated claims and the gathered evidence.

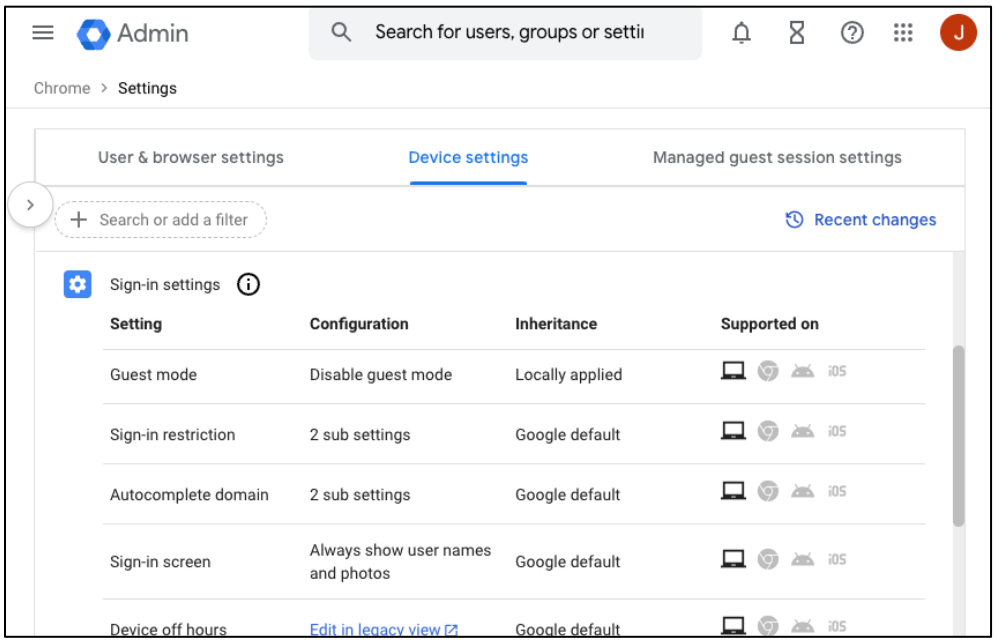
Enterprise and Education Lockdown

In enterprise and educational settings, the nature of device use presents unique security challenges. For enterprises, it's crucial to balance the need for employees to perform their tasks efficiently with the imperative to protect the organization's information and systems. Security measures must enable productivity while guarding against external and internal threats. In educational environments, the focus shifts towards safeguarding user privacy and minimizing the risk posed by insiders, given the higher likelihood of insider threats. This involves implementing controls to prevent students and staff from adversely affecting each other, as well as the institution's applications and systems.

> *In an enterprise or educational environment, the system can be locked down in a way that allows no alternative means of browsing the Internet to work around restrictions put in place by an organization (by actions that might include disabling crostini and disabling guest users or addition of non-enterprise users).*
>
> *Google ChromeOS Claim*

Validation

When a ChromeOS device undergoes Enterprise Enrollment, it can be managed and monitored by Enterprise administrators via the Google Admin console. This console provides a plethora of highly granular controls over the devices and accounts of enrolled users. While some degree of management and monitoring is available for Google accounts logged into other device types, such as iPhones, Android devices, and device-agnostic Chrome browsers, ChromeOS was the primary focus of this analysis and supports the highest degree of control.



When specifically examining the claim that "the system can be locked down in a way that allows no alternate means of browsing the Internet," Atredis Partners considered possible paths for accessing web content, which over the years has become a feature integrated with a huge array of services and applications. At the most basic level, as mentioned in the claim, if a user could log in as an unmanaged user or enable unmanaged Guest mode on their organization-owned device, all account-level restrictions would be circumvented; Guest mode is isolated from other accounts on the device and is not managed. There is a managed Guest mode feature, which allows for limits on login time and some other restrictions, but that is not within the scope of discussion here.

By default, any Google user can log into an enterprise-enrolled ChromeOS device, even if they are not a member of the organization; content/browsing restrictions will not apply to this user, but device-level restrictions, such as the ability to `powerwash` the device, will still apply. Administrators can limit which users can log into the device by explicitly defining a list of users and/or domains, where wildcards are accepted (e.g., `*@example.com`). Attempts to log in as other users are denied with the following message.

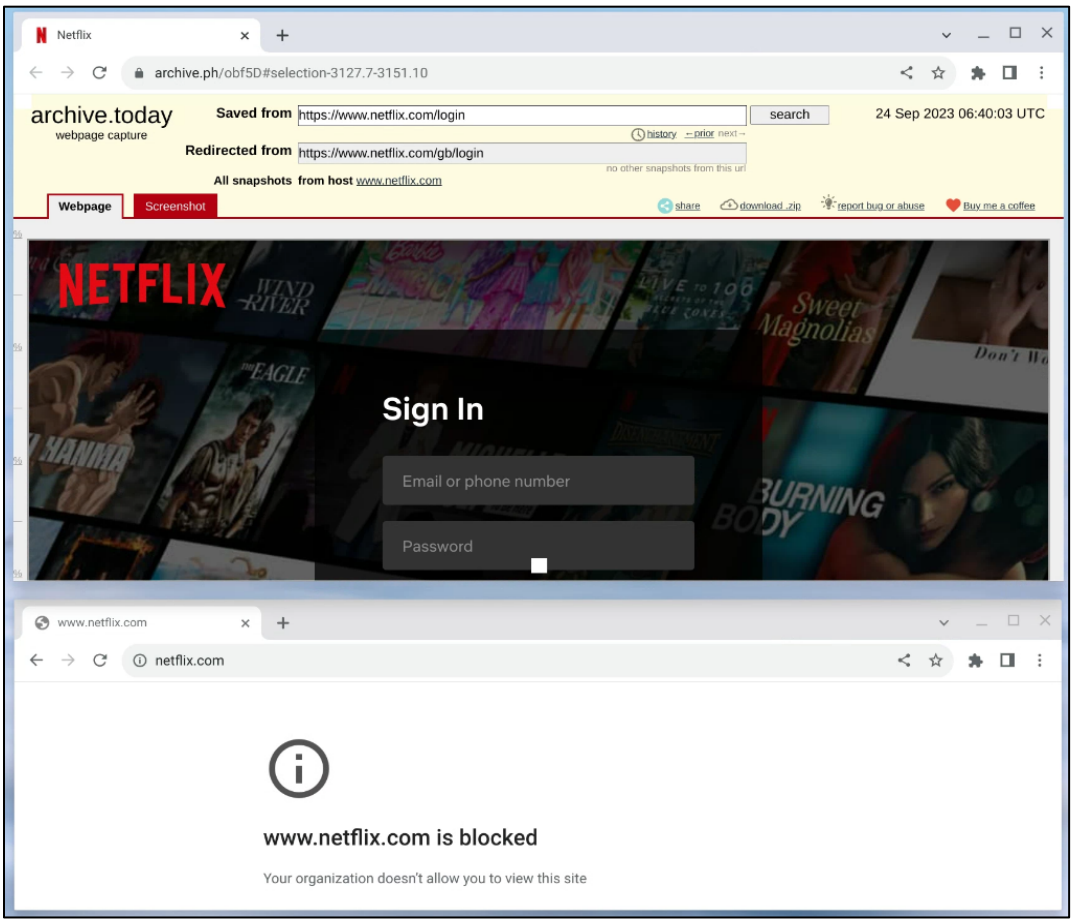
The granularity of control provided by the Admin Console extends to web content, including Chrome Extensions, which can be blocked depending on criteria like installation source, extension type, and the particular permissions requested. For example, an organization can disallow any extension that requests access to the device's microphone.

Where security and content are concerned, the degree of customization available is immense. Administrators have fine-grained controls over things like which sites can run JavaScript, set Cookies, load images, auto-play videos, auto-open downloaded files, request access to Bluetooth devices, etc.

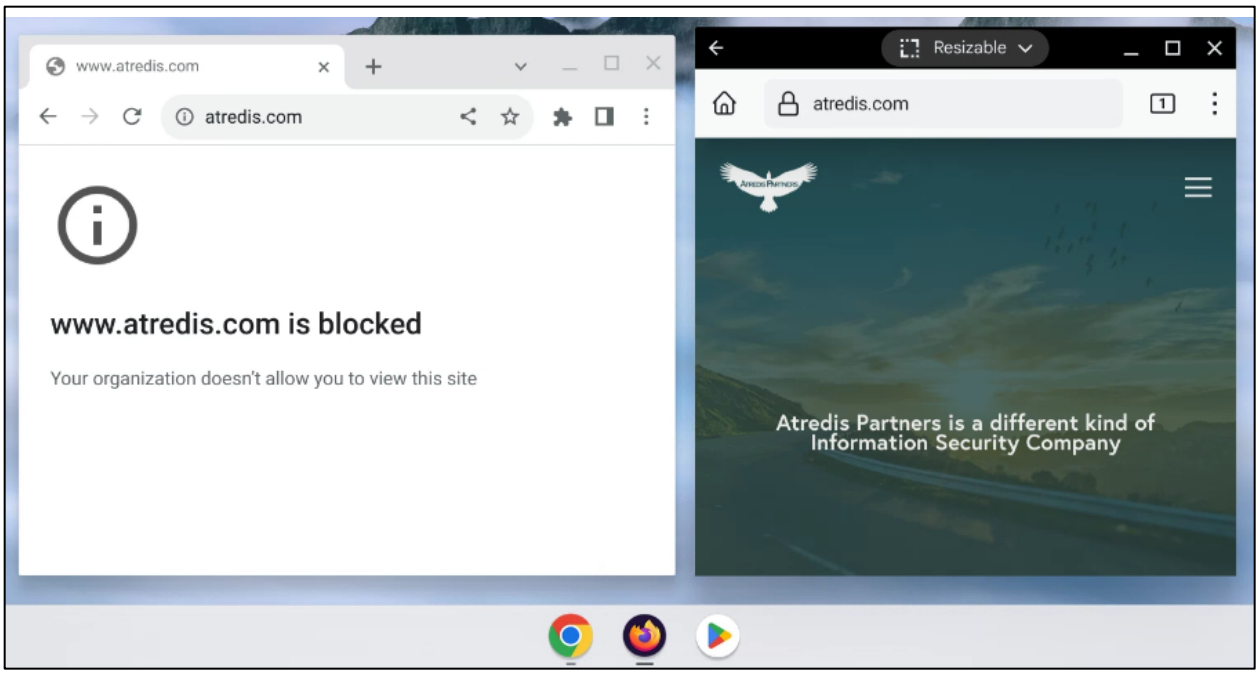
Atredis Partners found that when blocking domains, the documentation was helpful in explaining the various nuances, providing examples where needed. A subset of these examples can be found below.

Blocked URLs entry	Result
example.com	Blocks all requests to *example.com*, *www.example.com*, and *sub.www.example.com*
http://example.com	Blocks all HTTP requests to *example.com* and any of its subdomains, but allows HTTPS requests
https://*	Blocks all HTTPS requests to any domain
mail.example.com	Blocks requests to *mail.example.com* but not to *www.example.com* or *example.com*
.example.com	Blocks *example.com* but not its subdomains, like *example.com/docs*
.www.example.com	Blocks *www.example.com* but not its subdomains

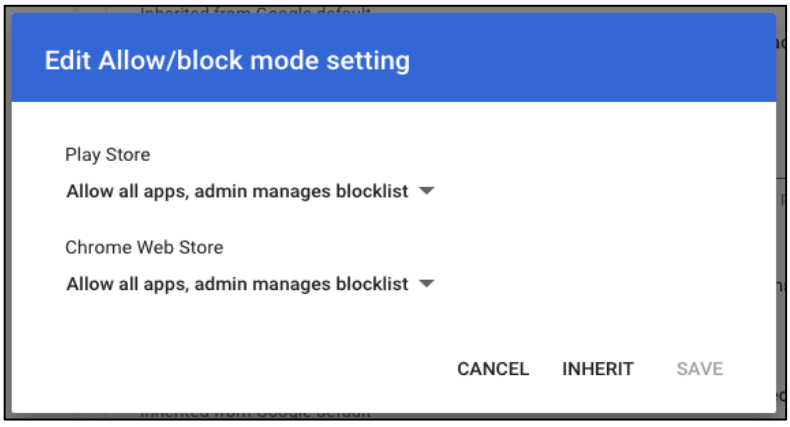
Atredis Partners attempted a few well-known methods for bypassing domain restrictions. For example, using an archive site, such as `archive.org` and `archive.ph` effectively bypassed the restriction. However, for some sites this does not provide a sufficiently robust and interactive experience.



ChromeOS support for installing Android applications drastically increases the available paths to accessing restricted content. Organization-defined URL restrictions do not apply to Android applications running within the ARCVM. This is demonstrated below, where the `atredis.com` domain is blocked on the host OS Chrome instance and allowed on the ARCVM Firefox application for Android.



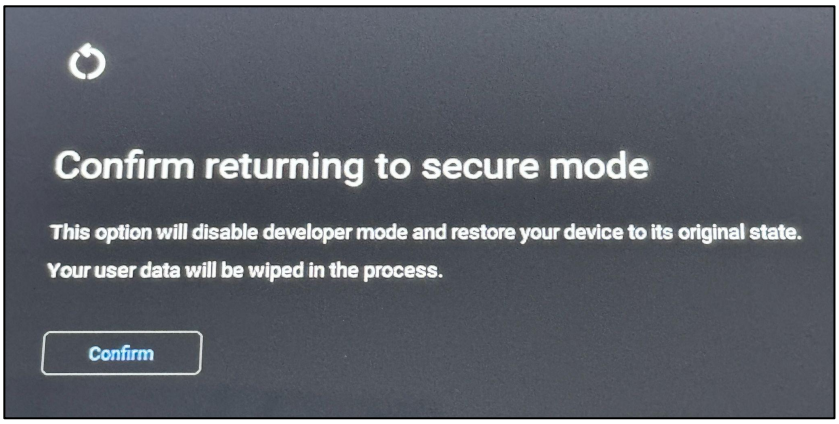
This limitation is outlined in the documentation for the URL blocking setting, along with detailed information on how to block URLs within Android apps. However, this configuration is only available for some apps; Atredis Partners found that these restrictions could not be applied to Web Browsers such as Firefox and Opera. Given the vast number of browsers available in the Play Store, the best approach would be to restrict Play Store access entirely, explicitly defining which applications should be allowed on user systems, and adding the configuration to block unintended URLs if there is any opportunity for the application to access the restricted web content.



Advanced features that provide more indirect methods for accessing web content, such as the Linux VM environment known as Crostini, can also be disabled. Atredis Partners found that when setting up a test Enterprise environment, Crostini was disabled by default. As expected, after enabling this feature, the restricted web content could be accessed via the command line with built-in tools such

as `curl` and `wget`. As noted in the Admin Console, blocking Linux VMs does not stop instances that are already running on user systems; one unexpected behavior noted by Atredis Partners was that once Linux VM usage was blocked, a user with the Linux VM already enabled could no longer remove the Linux development environment from their system. The feature had to be switched back to "Allowed" for the removal feature to have an effect.

To prevent users from subverting organization-wide security controls via Developer Mode, in addition to removing the ability to enter Developer Mode when already enrolled in an organization, Enterprise controls will force a Developer Mode device to return to secure mode immediately once the device is enrolled.



Note that this only applies when the device is fully enrolled on initial startup, not when an Enterprise-managed user logs into an unmanaged device. In the latter scenario, when a user logs into a Developer Mode system with their enterprise account, they can still access the underlying filesystem via Developer Mode. Atredis Partners identified the content policy file in `/home/root/{UID}/session_manager/policy/policy`. This binary file contained strings from the defined policy, including the list of blocked Play Store applications and URLs:

```
{"applications":[{"packageName":"com.opera.browser","installType":"BLOCKED"},{"pac
kageName":"org.mozilla.firefox","installType":"BLOCKED"}],"playStoreMode":"BLACKLI
ST","dpsInteractionsDisabled":true}
atredis.com
netflix.com
youtube.com
```

Atredis Partners edited this file to remove one of the domains; it was still not accessible via Chrome. After logging out, the enterprise user could no longer log in; the authentication process would simply loop. This avenue for exploitation was not further explored, as Developer Mode comes with numerous caveats about how system behavior can be affected.

To reduce the likelihood of users tampering with these policies to undermine the organization's security controls, administrators should have total control over the device during the provisioning and enrollment process, where the end-user only needs to log into the device with their managed Google account.

Comparative Analysis

macOS allows enterprise and education users to use third-party Mobile Device Management (MDM) solutions to control many aspects of the operating system and device. By using this, administrators can restrict what software is installed and can be installed. Limiting browsing to Safari is not a goal of these systems or a goal of macOS. There are applications that can be used to load resources from the Internet. iPadOS is more a direct comparison for educational and enterprise environments that require strict lockdown of users. MDM solutions can be used to effectively lockdown a device to only use applications that are suitable for students and the needs of the enterprise.

Windows 11 SE is a version of Windows that is designed for education use. By default, it ships with a limited number Win32 and Universal Windows Platform (UWP) applications, including Microsoft Edge. Administrators can use Microsoft Intune to make additional software available. This can effectively limit the ways that students can browse the Internet; however, some applications other than Microsoft Edge can load resources from the Web. For example, most Office applications can load media including photos and video.

In an enterprise environment, administrators can use fine-grained policies and a combination of MDM solutions to fully control a Windows 11 device.

Conclusion

Due to the number of applications available on macOS and Windows 11 (SE included) it is more likely that students and enterprise users could find ways to bypass controls dedicated to restricting use of the Internet to a single application. This is not a fair comparison to make given the language and context of the claim. A better comparison would be to compare the administrative expertise and associated costs required to effectively control a fleet of devices across an enterprise or educational environment. In this context, ChromeOS may provide the best solution given the limited configuration required to effectively lock down a device.

In reviewing the configuration controls available to administrators for ChromeOS, Atredis Partners did not identify any areas of improvement or discrepancies between the stated claims and the gathered evidence.

Seamless Updates

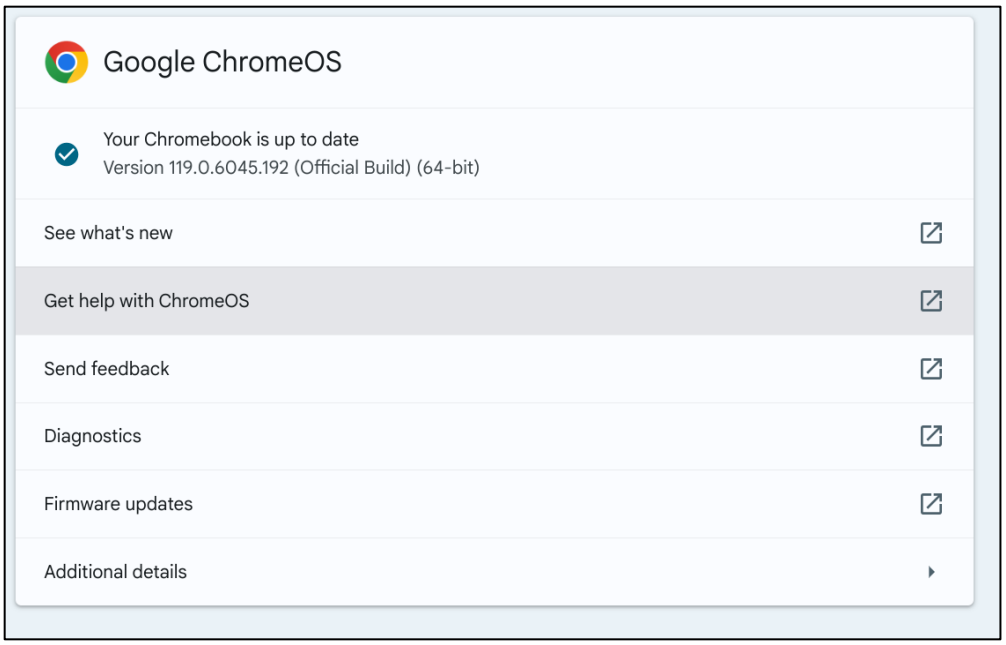
Operating systems, firmware, applications, and other software will inevitably experience security vulnerabilities. These vulnerabilities can be leveraged by attackers against users, exposing their sensitive data or making use of their system without their consent. Without a comprehensive patching and update solution system security can degrade over time, and the likelihood of exploitation increases.

> *Users should not have to worry about "patching" their systems for security bugs. Updates (including security updates) should be seamless.*
>
> *This means that maintaining your security posture is simple: reboot and you're up-to-date. ChromeOS is designed to boot quickly, which helps ensure staying up-to-date isn't a burden for the user.*
>
> *Google ChromeOS Claim*

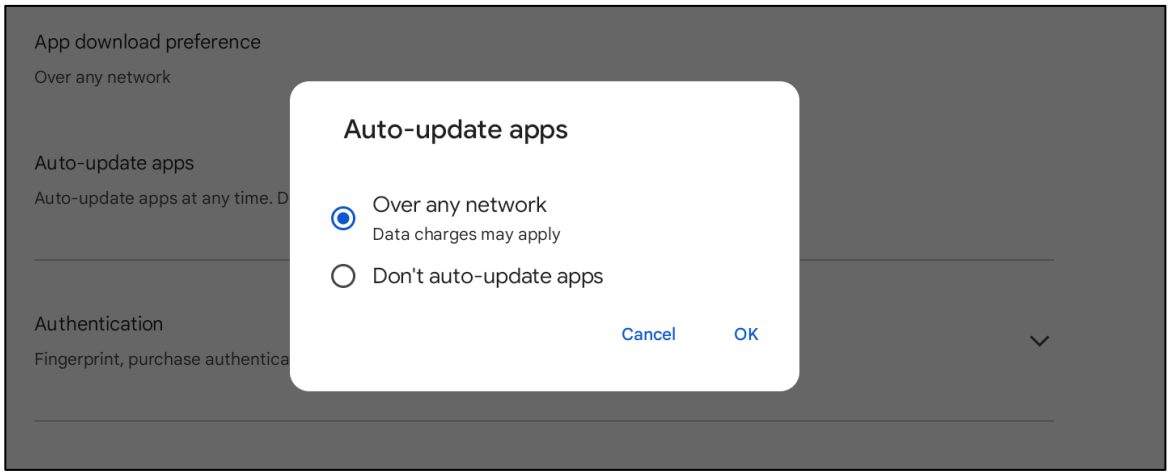
Validation

ChromeOS supports automatic updates by default. When ChromeOS has downloaded and installed updates that need a reboot, a notification will appear to the user in the bottom right corner stating "Update available." The user then selects "Restart" to update, and after reboot all updates are completed. System updates will update all system components due to the highly integrated nature of ChromeOS. Android applications or other VM isolated components may need separate updates.

Users can check for system updates manually using the Settings menu, which will also show if the user is up to date.



Android applications are updated from the Google Play Store and have their own update settings; by default, updates are performed and applied automatically.



Comparative Analysis

Windows 11 offers automatic updates for the operating system, device drivers, and Windows Defender definition files. Windows will attempt to install these updates as soon as they are downloaded. If a reboot is required to apply updates, Windows will attempt to reboot the system outside of the user's defined "Active hours" when the user is signed out. Users will also be shown a notification when interacting with the power button in the Start menu. Options are available to apply updates and reboot as soon as updates are available, as well as notify users when a reboot is required;

these options are disabled by default. Windows update can also manage updates for Microsoft Office; this is also disabled by default.

macOS also supports automatic system updates, and they are enabled by default. There are various additional settings to control updates including the following.

- Download new updates when available (Enabled by default)
    - Allows macOS to download updates as they are available instead of at time of install.
- Install macOS updates (Disabled by default)
    - Allows the system to install updates automatically.
- Install applications updates from the App Store (Disabled by default)
    - Allows the system to download and update applications installed from the App store.
- Install Security Responses and system files (Enabled by default)
    - Allows the system to apply security updates more rapidly as they are supplied from Apple. This is useful as they can be applied faster than the typical update process. Updates are specified by a letter following the macOS version (e.g. macOS 13.3.1 (a)).

In practice, some of these settings frequently do not behave as they stated, particularly the downloading and installation of updates. Many updates for macOS require accepting new Terms of Service and entering the user's administrator password.

Both macOS and Windows provide stores of applications and manage updates for installed software. Applications that are not installed through the Microsoft Store on Windows 11 or the App Store on macOS require their own update processes. Often these applications come with their own update services, use a package manager, check for updates when launched or at a defined interval, or require the users to check and apply them manually. These non-store applications are not managed by the operating system and may lack suitable update mechanisms.

Conclusion

The ChromeOS update experience is more seamless, requiring less user intervention than Windows 11 and macOS. This is due to simple settings with secure defaults, requiring minimal interaction to perform updates. Third-party applications on ChromeOS are managed by a store which also provides updates.

Atredis Partners did not identify any discrepancies between the claims made by Google and the experience validating this claim or using ChromeOS throughout the assessment.

Remote Access

Exposed network services increase the remote attack surface of an operating system. Any weakness in these services could provide a direct opportunity for an attacker to gain a foothold and potentially move laterally or escalate privileges. Along with the attack surface, it is vital to examine the security controls available to limit network access and functionality, the ease-of-use of these controls, and the operating system's general design of network services to ensure the system provides a strong framework for protecting the user from unwittingly exposing the network to potential attacks.

> *Attackers can't access ChromeOS devices remotely by default, users have to explicitly allow Android applications specific access (for example, a VPN).*
>
> *Google ChromeOS Claim*

Validation

ChromeOS uses iptables to configure the Linux kernel `Netfilter` subsystem as a firewall. The rules deploy a default deny policy for ingress, and only ICMP, Simple Service Discovery Protocol (SSDP), and multicast (mDNS) are allowed. SSDP and mDNS are protocols used for device discovery and when attempting to identify fileshares on a local network.

The user is not able to modify these rules directly, and there is no ability to limit egress connections. This was validated using iptables from a developer console; additionally, traditional network scanning tools confirmed that there was no default remote access.

```
$ iptables -L -v -n
Chain INPUT (policy DROP 188 packets, 7160 bytes)
 pkts bytes target     prot opt in  out    source              destination
 131K  352M ACCEPT     all  --  *   *      0.0.0.0/0           0.0.0.0/0          state
RELATED,ESTABLISHED
  165 10822 ACCEPT     all  --  lo  *      0.0.0.0/0           0.0.0.0/0
    0     0 ACCEPT     icmp --  *   *      0.0.0.0/0           0.0.0.0/0
 1327  279K ACCEPT     udp  --  *   *      0.0.0.0/0           224.0.0.251        udp dpt:5353
  286 59638 ACCEPT     udp  --  *   *      0.0.0.0/0           239.255.255.250    udp dpt:1900
```

Service discovery protocols are often used by attackers to poison requests and coerce systems into connecting to resources they control. During review of the attack surface and susceptibility of these attacks, it was identified that the NetBIOS Name Service (NBNS) may be used when attempting to mount a fileshare; however, poisoned (and all) responses are not allowed by the firewall. mDNS is susceptible provided the user is using the `.local` TLD when using Chrome. Even then, poisoning is dependent on timing between an attacker and a legitimate device, and mDNS is not used when adding new fileshares, reducing the likelihood of successful attacks against mDNS.

Two additional ChromeOS features theoretically have the potential to expose services. The first is ChromeOS support for Android applications. These applications are installed and executed using the

Android Runtime on ChromeOS (ARC) paradigm. The latest version, ARCVM, uses a virtual machine (VM) on top of Kernel Virtual Machine (KVM). These applications may create listening network services, but they are not available over the network.

Second is Linux developer mode, which creates a VM where users can run Linux applications and access a terminal. Network services inside of this VM can be exposed to the network using the Port forwarding UI. This creates multiple iptables rules that modify the `INPUT` and `FORWARDING` chains.

Comparative Analysis

macOS provides a built-in firewall that is disabled by default. The Application Firewall (ALF) differs from a traditional network firewall that uses IP addresses, ports, and protocols in that it operates at the application layer and uses process identifiers to control access on a per application basis. The firewall can block incoming connections but has no capability for limiting outgoing connections. Third-party solutions or `pfctl` (a tool used to control an additional firewall) are required for this. Depending on the profile deployed, applications may be allowed to accept incoming connections. There are no services exposed by default that enable remote access; however, applications and system services may expose themselves to the network, which increases the attack surface.

Windows 11 uses the Microsoft Defender Firewall to filter incoming and outgoing network connections. The firewall is enabled by default and comes configured with multiple profiles. Rules are separated into Inbound and Outbound groups and can be configured by a user in numerous ways. These included multiple UI options as well as command line utilities. It is not typical for a user to directly interact with the firewall and create rules. Instead, when an application attempts to listen for an incoming connection for the first time, the firewall will prompt the user and ask if they would like to allow it, and a new rule is created. More commonly, an application's installer will modify the firewall rules directly. Depending on the profile used, services that can be used by attackers to facilitate remote access may be exposed. The default policy does not allow for Network discovery or File and printer sharing. Additionally, common protocols used to identify a live system such ICMP are also not enabled. Network discovery allows for the use of multiple broadcast protocols to identify a device on a network. File and printer sharing exposes NetBIOS and Server Message Block (SMB) services to the network. Historically these present a large attack surface and, in a domain environment, are common targets.

Both macOS and Windows 11 are susceptible to poisoning via various protocols, including mDNS and NBNS. In practice, Windows is the most targeted platform in this regard, this is due to the availability of public tools and the impact of a successful attack. NBNS and Link-Local Multicast Name Resolution (LLNR) are particularly susceptible. mDNS has the same constraints as attacks described against ChromeOS.

Conclusion

Due to default behavior, the abundance of configuration options, and the ability for applications to modify behavior, ChromeOS emerges as the most effective solution at limiting remote attack surface.

No discrepancies were identified between the ChromeOS claim and testing performed. An area of improvement for the ChromeOS team would be to remove support for NBNS completely. However, this may be difficult as it is built into underlying components used by the platform.

Appendix I: About Atredis Partners

Atredis Partners was created in 2013 by a team of security industry veterans who wanted to prioritize offering quality and client needs over the pressure to grow rapidly at the expense of delivery and execution. We wanted to build something better, for the long haul.

In six years, Atredis Partners has doubled in size annually, and has been named three times to the Saint Louis Business Journal's "Fifty Fastest Growing Companies" and "Ten Fastest Growing Tech Companies". Consecutively for the past three years, Atredis Partners has been listed on the Inc. 5,000 list of fastest growing private companies in the United States.

The Atredis team is made up of some of the greatest minds in Information Security research and penetration testing, and we've built our business on a reputation for delivering deeper, more advanced assessments than any other firm in our industry.

Atredis Partners team members have presented research over forty times at the BlackHat Briefings conference in Europe, Japan, and the United States, as well as many other notable security conferences, including RSA, ShmooCon, DerbyCon, BSides, and PacSec/CanSec. Most of our team hold one or more advanced degrees in Computer Science or engineering, as well as many other industry certifications and designations. Atredis team members have authored several books, including *The Android Hacker's Handbook*, *The iOS Hacker's Handbook*, *Wicked Cool Shell Scripts*, *Gray Hat C#*, and *Black Hat Go*.

While our client base is by definition confidential and we often operate under strict nondisclosure agreements, Atredis Partners has delivered notable public security research on improving security at Google, Microsoft, The Linux Foundation, Motorola, Samsung and HTC products, and were the first security research firm to be named in Qualcomm's Product Security Hall of Fame. We've received four research grants from the Defense Advanced Research Project Agency (DARPA), participated in research for the CNCF (Cloud Native Computing Foundation) to advance the security of Kubernetes, worked with OSTIF (The Open Source Technology Improvement Fund) and The Linux Foundation on the Core Infrastructure Initiative to improve the security and safety of the Linux Kernel, and have identified entirely new classes of vulnerabilities in hardware, software, and the infrastructure of the World Wide Web.

In 2015, we expanded our services portfolio to include a wide range of advanced risk and security program management consulting, expanding our services reach to extend from the technical trenches into the boardroom. The Atredis Risk and Advisory team has extensive experience building mature security programs, performing risk and readiness assessments, and serving as trusted partners to our clients to ensure the right people are making informed decisions about risk and risk management.