How to boot Ubuntu on a Cr-48

Introduction

While Chrome OS verified boot protects against unintended system modification by malicious or buggy software, **the ability to hack your own device is an intentional design feature of Google Chrome notebooks**. The instructions for building your own version of Chromium OS, and installing it on a Cr-48 are given elsewhere. Some enthusiasts, however, may want to install something completely different. This page provides an example, showing how the official Chrome OS software can coexist with Ubuntu, a popular linux distribution. I'm assuming that you're already somewhat familiar with the Chromium OS development environment.

Caution: Modifications you make to the system are not supported by Google, may cause hardware, software or security issues and may void warranty. Be very careful.

These instructions work on the Cr-48 as originally shipped. There are several magic numbers that may change following software updates or on other Chromium OS-based machines. Even if the numbers don't exactly match, the basic idea is the same.

In the examples below:

```
The Cr-48 shell is formatted like this.
```

```
The Chromium OS build chroot is formatted like this.
```

```
The standard linux workstation shell is formatted like this.
```

Fair warning

This is kind of a cheat. The Cr-48's boot process does not support `initrd`, which is required by Ubuntu. That leaves us three possibilities:

1. Use the existing Chrome OS kernel with the Ubuntu rootfs.
2. Recompile the Ubuntu kernel to do without `initrd` (ugh).
3. Modify the Chrome OS bootstub to handle `initrd` (double-ugh).

Let's take door #1.

Free up some SSD space for Ubuntu

To begin our journey, first switch the Cr-48 into developer mode and reboot. When you see the blue frowny face with the "Chrome OS verification is turned off" message, either wait 30 seconds or hit Ctrl-D to boot immediately. This screen is always shown when booting in developer mode, to ensure that someone doesn't change your OS without your knowledge.

Switch to VT2 (press [Ctrl] [Alt] [=>]), and log in as user 'chronos' (no password required), then run `sudo bash`.

To prevent the lock screen from logging you out of VT2, leave Chrome OS parked on the login screen. To prevent the backlight from dimming while in VT2, run `sudo initctl stop powerd`.

Take a look at the SSD layout. It should be something like this (the UUID values will all be different, of course):

```
localhost ~ # cgpt show /dev/sda
     start      size    part  contents
         0         1          PMBR (Boot GUID: 045E5C92-6F57-9B46-BDCC-99BFF876E469)
         1         1          Pri GPT header
         2        32          Pri GPT table
    266240  22622208       1  Label: "STATE"
                              Type: Linux data
                              UUID: 1844A16D-AEC9-7C4B-9553-C3EB4814F6BB
      4096     32768       2  Label: "KERN-A"
                              Type: ChromeOS kernel
                              UUID: D176DC60-81F1-654E-8953-E3D28019738C
                              Attr: priority=3 tries=0 successful=1
  27082752   4194304       3  Label: "ROOT-A"
                              Type: ChromeOS rootfs
                              UUID: 0193CA51-DA12-9847-A715-C90433E55F60
     36864     32768       4  Label: "KERN-B"
                              Type: ChromeOS kernel
                              UUID: F1A2C65C-CC22-FF4A-A8BC-67BA233F3D40
                              Attr: priority=0 tries=15 successful=0
  22888448   4194304       5  Label: "ROOT-B"
                              Type: ChromeOS rootfs
                              UUID: B3361FB5-4DAC-9344-B7E5-870B7AC5FEA1
        34         1       6  Label: "KERN-C"
                              Type: ChromeOS kernel
                              UUID: B6954485-4295-9749-956A-C315B01FB684
```

```
                           Attr: priority=0 tries=15 successful=0
       35         1       7  Label: "ROOT-C"
                           Type: ChromeOS rootfs
                           UUID: 5B5202B5-F74B-714E-9538-ADE56B2E5662
    69632     32768       8  Label: "OEM"
                           Type: Linux data
                           UUID: 84971802-0D1C-504B-9CB5-DEA896F0AD3F
       36         1       9  Label: "reserved"
                           Type: ChromeOS reserved
                           UUID: 77375DA6-8F07-704A-BBF4-2BCA662BFDFF
       37         1      10  Label: "reserved"
                           Type: ChromeOS reserved
                           UUID: 6880F478-EB05-8B4B-B951-94A96076263E
       38         1      11  Label: "reserved"
                           Type: ChromeOS reserved
                           UUID: 12DDC236-8FDF-4049-9A2D-10FAB17D3AA9
   233472     32768      12  Label: "EFI-SYSTEM"
                           Type: EFI System Partition
                           UUID: 045E5C92-6F57-9B46-BDCC-99BFF876E469
 31277199        32          Sec GPT table
 31277231         1          Sec GPT header
```

The units are 512-byte disk sectors.

On Chrome OS there are several bootable images, each composed of a kernel and a root filesystem (rootfs). For a given image, the kernel and rootfs are stored on a pair of consecutive partitions. Two of these images, Image-A and Image-B, are for official Chrome OS:

- KERN-A and ROOT-A on partitions `/dev/sda2` and `/dev/sda3`
- KERN-B and ROOT-B on partitions `/dev/sda4` and `/dev/sda5`

When the Chrome OS autoupdater downloads a new image (every six weeks or so, as new versions are pushed out), it alternately stores it in Image-A and Image-B - whichever image isn't currently running. The autoupdater even runs in developer mode, as long as we are running an official Chrome OS image.

For Ubuntu, we'll use the currently unassigned Image-C, composed of KERN-C and ROOT-C on partitions `/dev/sda6` and `/dev/sda7`, respectively.

However, initially they are too small so, first we need to grow them by stealing from the upper half of the stateful partition, STATE (`/dev/sda1`). Initially, STATE is 22622208 512-byte sectors, or just under 11 GB. From this we'll set aside 16 MB for KERN-C, and 5 GB for ROOT-C.

```
umount /mnt/stateful_partition
cgpt add -i 1 -b 266240    -s 12103680 -l STATE   /dev/sda
cgpt add -i 6 -b 12369920  -s 32768    -l KERN-C  /dev/sda
cgpt add -i 7 -b 12402688  -s 10485760 -l ROOT-C  /dev/sda
```

Using these values ensures that only the original STATE partition is affected.

Now the partition table should look something like this (changes in **bold**):

```
localhost ~ # cgpt show /dev/sda
     start      size    part  contents
         0         1          PMBR (Boot GUID: 045E5C92-6F57-9B46-BDCC-99BFF876E469)
         1         1          Pri GPT header
         2        32          Pri GPT table
    266240  12103680       1  Label: "STATE"
                           Type: Linux data
                           UUID: 1844A16D-AEC9-7C4B-9553-C3EB4814F6BB
      4096     32768       2  Label: "KERN-A"
                           Type: ChromeOS kernel
                           UUID: D176DC60-81F1-654E-8953-E3D28019738C
                           Attr: priority=3 tries=0 successful=1
  27082752   4194304       3  Label: "ROOT-A"
                           Type: ChromeOS rootfs
                           UUID: 0193CA51-DA12-9847-A715-C90433E55F60
     36864     32768       4  Label: "KERN-B"
                           Type: ChromeOS kernel
                           UUID: F1A2C65C-CC22-FF4A-A8BC-67BA233F3D40
                           Attr: priority=0 tries=15 successful=0
  22888448   4194304       5  Label: "ROOT-B"
                           Type: ChromeOS rootfs
                           UUID: B3361FB5-4DAC-9344-B7E5-870B7AC5FEA1
  12369920     32768       6  Label: "KERN-C"
                           Type: ChromeOS kernel
                           UUID: B6954485-4295-9749-956A-C315B01FB684
                           Attr: priority=0 tries=15 successful=0
  12402688  10485760       7  Label: "ROOT-C"
                           Type: ChromeOS rootfs
                           UUID: 5B5202B5-F74B-714E-9538-ADE56B2E5662
     69632     32768       8  Label: "OEM"
                           Type: Linux data
```

```
                          UUID: 84971802-0D1C-504B-9CB5-DEA896F0AD3F
        36         1       9  Label: "reserved"
                          Type: ChromeOS reserved
                          UUID: 77375DA6-8F07-704A-BBF4-2BCA662BFDFF
        37         1      10  Label: "reserved"
                          Type: ChromeOS reserved
                          UUID: 6880F478-EB05-8B4B-B951-94A96076263E
        38         1      11  Label: "reserved"
                          Type: ChromeOS reserved
                          UUID: 12DDC236-8FDF-4049-9A2D-10FAB17D3AA9
    233472     32768      12  Label: "EFI-SYSTEM"
                          Type: EFI System Partition
                          UUID: 045E5C92-6F57-9B46-BDCC-99BFF876E469
  31277199        32          Sec GPT table
  31277231         1          Sec GPT header
```

At this point we should destroy the STATE partition contents, because otherwise at the next boot the startup scripts may notice that it's corrupted and will erase it using the old size (obtained from dumpe2fs) and not the new one. Best to be safe and just zap it now. This will take a couple of minutes to run:

```
dd if=/dev/zero of=/dev/sda bs=131072 seek=1040 count=47280
```

To speed up SSD access, we use a block size of 131,072 bytes (128 KB), which aligns with the SSD erase size.

And now reboot so that the startup script recreates the required files in the stateful partition. This will do a safe wipe first, so you'll have to wait a bit (again).

At this point we're through fiddling with the partition table, so you may want to go back to VT2 and set a password according to these instructions.

Acquire an Ubuntu filesystem

First, although the Cr-48 CPU and BIOS is 64-bit, the kernel and rootfs are presently 32-bit. Since we're reusing the Chrome OS kernel, we'll download and use the 32-bit ubuntu-10.10-desktop-i386.iso from an Ubuntu CD mirror. For example:

```
wget http://mirror.anl.gov/pub/ubuntu-iso/CDs/10.10/ubuntu-10.10-desktop-i386.iso
```

Second, even in developer mode, the Cr-48 will only boot external drives that are signed by Google. That means we can't (easily) use the normal Ubuntu live installer. Instead, I'll create an empty disk image file with a 5G partition on it, and use VirtualBox to install into that. Then I'll transfer that raw partition onto the Cr-48.

To install VirtualBox (and its command line management tool, `VBoxManage`) on Ubuntu:

```
sudo aptitude install virtualbox-ose
```

I'll do this on my normal linux workstation, of course. I'll create the file inside the chroot so I can use the cgpt tool, but I'll run VirtualBox from outside. We could create the file using fdisk, but using cgpt ensures all the numbers match. You could also build the cgpt tool for use external to the chroot, but whatever.

We'll need our disk image to be a little larger than 5G so it has room for the GPT headers.

So, do the following in the Chromium OS chroot:

```
cd /tmp
rm -rf test.bin
dd if=/dev/zero bs=512 seek=10486015 count=1 of=test.bin
cgpt create test.bin
cgpt boot -p test.bin
cgpt add -b 128 -s 10485760 -t data -l ubuntu test.bin
```

From outside the chroot, translate the binary into a virtualbox disk:

```
cd /path/to/chromiumos/chroot/tmp
VBoxManage convertdd test.bin test.vdi --format VDI
```

Launch the VirtualBox GUI (`virtualbox`) and create a new virtual machine with 2G of RAM. Configure `test.vdi` as the existing hard disk (SATA port 0), and attach the Ubuntu .iso as the IDE secondary master. Everything else can be left as default.

Boot the virtual machine and install Ubuntu onto `/dev/sda1`. The Chrome OS kernel does not use swap memory, so be sure to specify partitions

manually. Select the middle `/dev/sda1`, and format it for `ext2`, and mount on `/`. The ubuntu installer should warn you that you have no swap partition, and it will therefore disable swap. This is what we want.

Once Ubuntu is installed, working and updated, shut the virtual machine down and convert the `.vdi` image back into a binary:

```
VBoxManage internalcommands converttoraw test.vdi test_new.bin
```

Now, we'll extract just the Ubuntu rootfs partition from the binary file:

```
dd if=test_new.bin bs=512 skip=128 count=10485760 of=rootfs.bin
```

Next, we copy the ubuntu rootfs directly to the Cr-48 ROOT-C partition (`/dev/sda7`). To copy the file, run the following command on the Cr-48, where USER@HOST refers to the account and host from which you are copying.

Note: the root filesystem is a huge file that will take quite a while to upload. This is much less painful with a fast network connection, so, if possible, use a USB-to-ethernet adapter to avoid going through Wi-Fi.

```
ssh USER@HOST cat /path/to/rootfs.bin | dd of=/dev/sda7
```

If you have to use Wi-Fi, you can reduce the wait with a bit of compression (thanks to Jay Lee for pointing this out):

```
ssh USER@HOST bzip2 -c /path/to/rootfs.bin | bunzip2 -c | dd of=/dev/sda7
```

If your network is flaky you might find it easier to use `scp` to copy the compressed image to the Cr-48 first, then uncompress and write to `/dev/sda7`. Or you could just copy the `rootfs.bin` file to a USB stick instead of using the network, but it's likely to take about as long. You always want to use `dd` to copy the rootfs image into `/dev/sda7`, of course.

You can see that it worked by mounting the new rootfs and looking around:

```
mkdir /tmp/urfs
mount /dev/sda7 /tmp/urfs
ls /tmp/urfs
```

While we're looking, let's copy the `cgpt` tool into Ubuntu's rootfs. Make sure it's executable. We'll need this later.

```
cp /usr/bin/cgpt /tmp/urfs/usr/bin/
chmod a+rx /tmp/urfs/usr/bin/cgpt
```

We'll need to copy all the kernel modules too.

```
cd /lib/modules
cp -ar * /tmp/urfs/lib/modules/
```

That should do it.

```
umount /tmp/urfs
```

Configure the kernel

The next step is to copy the currently running Chrome OS kernel to KERN-C (`/dev/sda6`). Remember, an image consists of a kernel and a rootfs on consecutive partitions. So, to figure out which kernel is running, we can check which partition is currently mounted as the rootfs:

```
rootdev -s
```

If this prints `/dev/sda3` (ROOT-A), then our kernel is on `/dev/sda2` (KERN-A).
If this prints `/dev/sda5` (ROOT-B), then our kernel is on `/dev/sda4` (KERN-B).
Assuming our kernel is on `/dev/sda2` (KERN-A), copy it to `/dev/sda6` (KERN-C):

```
dd if=/dev/sda2 of=/dev/sda6
```

Now we need to change the kernel command line to use our Ubuntu rootfs instead of a Chrome OS rootfs. For this, we'll use `make_dev_ssd.sh`,

located in the Chromium OS source tree under `src/platform/vboot_reference/scripts/image_signing/`. The latest version has options to change individual kernel command lines.

Use `scp` to copy it from your host to the stateful partition of the Cr-48. Note we'll also need `common_minimal.sh` from the same directory.

```
cd /mnt/stateful_partition
scp USER@HOST:/some/path/to/latest/make_dev_ssd.sh .
scp USER@HOST:/some/path/to/latest/common.sh .
```

Extract the existing kernel command line from KERN-C, and save it to a file named `foo.6` (the .6 extension is added by the script):

```
sh ./make_dev_ssd.sh --partitions '6' --save_config foo
```

The default Chrome OS kernel command line looks something like this:

```
quiet console=tty2 init=/sbin/init add_efi_memmap boot=local rootwait ro noresume noswap i915.modeset=1
loglevel=1 cros_secure kern_guid=%U tpm_tis.force=1 tpm_tis.interrupts=0 root=/dev/dm-0
dm_verity.error_behavior=3 dm_verity.max_bios=-1 dm_verity.dev_wait=1 dm="vroot none ro,0 1740800 verity
/dev/sd%D%P /dev/sd%D%P 1740800 1 sha1 50adbfb72bb1efda0c1a86dcd1c1d6a0b46726d1" noinitrd
```

The only editor on the Cr-48 is qemacs, so open foo.6 with qemacs:

```
qemacs foo.6
```

Edit the file to look like this:

```
console=tty1 init=/sbin/init add_efi_memmap boot=local rootwait ro noresume noswap i915.modeset=1 loglevel=7
kern_guid=%U tpm_tis.force=1 tpm_tis.interrupts=0 root=/dev/sda7 noinitrd
```

Then save (Ctrl-x Ctrl-s) and exit (Ctrl-x Ctrl-c). qemacs occasionally gets confused, so double-check to be sure you have the right content:

```
cat foo.6
```

Finally, use `make_dev_ssd.sh` to replace the kernel command line in KERN-C:

```
sh ./make_dev_ssd.sh --partitions '6' --set_config foo
```

The kernel command line is part of the kernel partition, and the entire partition must be cryptographically signed in order to work. We can't replace the command line part without affecting the entire kernel partition. The script will save a backup copy before replacing the partition content, but we shouldn't need it.

Set boot priority

At this point we should have the following situation:

Image-A is an official Google Chrome OS which can boot either in normal mode or dev-mode.
Image-B is (or will be after the first autoupdate) another official Google Chrome OS which can boot either in normal mode or dev-mode.
Image-C is Chrome OS kernel with a modified command line and an Ubuntu rootfs, which can only boot in dev-mode.

Next, we adjust the priority among the images so we can try out our Ubuntu image. The image priority is an attribute of its kernel partition. Run `cgpt show /dev/sda` again, to see the kernel priorities:

```
localhost ~ # cgpt show /dev/sda
...
      4096     32768       2  Label: "KERN-A"
                              Type: ChromeOS kernel
                              UUID: D176DC60-81F1-654E-8953-E3D28019738C
                              Attr: priority=3 tries=0 successful=1
...
     36864     32768       4  Label: "KERN-B"
                              Type: ChromeOS kernel
                              UUID: F1A2C65C-CC22-FF4A-A8BC-67BA233F3D40
                              Attr: priority=0 tries=15 successful=0
...
  12369920     32768       6  Label: "KERN-C"
                              Type: ChromeOS kernel
                              UUID: B6954485-4295-9749-956A-C315B01FB684
```

```
                                Attr: priority=0 tries=15 successful=0
```

The priority determines the order in which the BIOS tries to find a valid kernel (bigger is higher, zero means don't even try). The tries field is decremented each time the BIOS tries to boot it, and if it's zero, the kernel is considered invalid (this lets us boot new images without looping forever if they don't work). The successful field overrides the tries field, but is only set by the OS once it's up and running.

Let's change the priority of KERN-C to 5:

```
cgpt add -i 6 -P 5 -T 1 -S 0 /dev/sda
```

This makes KERN-C the highest priority, but only gives us one chance to boot it. That way if it doesn't work, we're not completely stuck.

If you reboot now, you should come up in Ubuntu! Note that Computer Science Standard Answer #1 applies: It Works For Me™

If something went wrong and Ubuntu crashes or you powered off, the tries field for KERN-C will have been decremented to 0 and you'll fall back to booting Chrome OS.

Assuming that Ubuntu booted and you could log in, go to Applications->Accessories->Terminal to get a shell, and run

```
sudo cgpt add -i 6 -P 5 -S 1 /dev/sda
```

This will mark the Ubuntu kernel as valid, so it will continue to boot next time.

Now you can switch back and forth between the official Chrome OS release and Ubuntu just by flipping the dev-mode switch. Going from dev-mode to normal mode erases STATE (`/dev/sda1`), but much more quickly. Going from normal to dev-mode again would normally do a slow erase of `/dev/sda1`, but since we're booting Ubuntu that doesn't happen.

This works because although KERN-C has the highest priority, it isn't signed by Google. In dev-mode that's okay, but in normal mode it will be rejected by the BIOS. Since we've set the successful flag to 1, the BIOS won't mark it invalid but will just skip it each time. This makes the normal-mode boot time slightly longer, but only by a second or two.

Of course you could also switch between images from within dev-mode just by manually setting the priorities with `cgpt` before rebooting.

Note that if the normal image autoupdates, it will probably change the kernel priorities so that Image-C is no longer the highest and the next time you switch to dev mode, you will
 a) have a long wait,
 b) still be running Chrome OS, and
 c) have to use cgpt to raise the KERN-C priority again.

But, because autoupdate only switches between Image-A and Image-B, the Ubuntu kernel and rootfs shouldn't be affected.

Footnotes

Ugh

Recompiling the Ubuntu kernel should be possible, with a few caveats:

First, you'll have to modify or reconfigure the kernel to not use initrd. initramfs should work though.

Second, official Chrome OS still has a few closed-source hardware drivers. The pure open-source Chromium OS experience is therefore a bit sub-optimal, although still quite useable.

Third, if you use the 32-bit image, you'll also have to modify the kernel source to boot correctly from the 64-bit Chrome OS BIOS.
See http://git.chromium.org/cgi-bin/gitweb.cgi?p=kernel.git;a=commit;h=a7cfa1075c04df162d58dc2ed93df6045e9c3271 for details.

If you use a 64-bit image it shouldn't need the boot hacks, but driver support is much less robust.

Finally, you'll have to sign the kernel yourself and put it into the KERN-C partition. That is actually the easiest step, but it's not really well documented. Look in the developer pages.

Double-ugh

`initrd` is typically handled by the bootloader, which reads the specified image from the disk into RAM and passes the address to the kernel as it's invoked.

The Chrome OS BIOS is a modified EFI BIOS. The bootstub is a standard EFI Application, but it's embedded in the kernel image in a dedicated partition type, rather than accessible through a FAT filesystem. To decrease boot time, the BIOS does not discover or pass the standard disk drive handles to the bootstub, so the bootstub doesn't know anything about disks or filesystems. There is also no Compatibility Support Module in the BIOS. In theory `elilo` or `grub2` could replace the bootstub, but they would have to reimplement some of the device discovery functions normally done by an EFI BIOS.

If you want to take this on, go for it. That would let us create a kernel partition that just contained an EFI bootloader, which could then chain-boot to external USB drives, etc. That might be kind of cool.

Some clarification in response to the wharrgarbl resulting from this howto...

- The dev-mode switch is purely for hackers who want to jailbreak their notebook. The people who build the notebook don't need it (duh).
- Things are locked down by default to try to prevent Bad People from rooting your device without your knowledge. It's always been intended that you should be able to root it yourself. Go read the design document for details and background.
- It's not a trick. We really don't care what you do with your own property. As long as you don't crack open the case, you should always be able to restore it to the original state.
- I put this hack together in about a day (with another day or so for my coworkers to proofread it) which is why it's so complicated. I figured you'd want to see an example sooner rather than later.
- wfrichar@chromium.org wrote this. I work for Google, on the Chrome OS verified boot stuff.