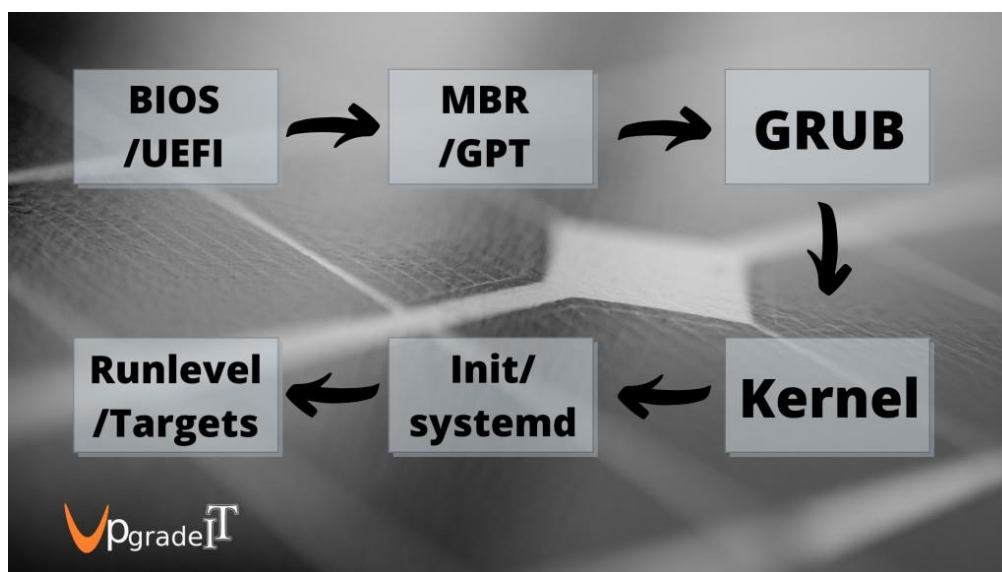


آزمایش ۲ - آشنایی با مراحل بوت شدن لینوکس و اعمال تغییرات و کامپایل مجدد هسته‌ی سیستم عامل

۲.۱ مراحل بوت شدن لینوکس



مراحل بوت شدن، با روشن کردن کلید پاور در کامپیوتر و یا restart آغاز می‌گردد. در مرحله‌ی اول، دستوالعمل‌های ذخیره شده، در BIOS یا UEFI اجرا می‌گردد. در مادربردهای امروزی از UEFI به جای BIOS استفاده می‌گردد. گرچه عموماً، از هر دو گزینه BIOS و UEFI پشتیبانی می‌شود.

مرحله‌ی اول BIOS/UEFI

BIOS

BIOS، نوعی firmware یا میان‌افزار می‌باشد. اولین برنامه‌ای که، پس از روشن شدن کامپیوتر، اجرا می‌گردد، BIOS یا Basic Input Output system است. BIOS، در چپ‌ی بر روی مادربرد قرار دارد. وظیفه‌ی BIOS، چک کردن اولیه‌ی سخت‌افزار کامپیوتر است. به این عمل POST یا Power On Self Test می‌گویند. پس از چک کردن سخت‌افزار BIOS به سراغ دیسک فعال رفته و سکتور بوت دیسک را، در حافظه‌ی اصلی قرار داده و اجرا می‌کند. BIOSهای قدیمی، بر روی حافظه‌ی ROM ذخیره می‌شدند. امروزه BIOS، بر روی حافظه‌ی Flash ذخیره می‌گردد. مزیت ذخیره بر روی حافظه‌ی فلش، قابلیت بروزرسانی آن می‌باشد. از معایب آن، خطر آلوده شدن BIOS به Rootkit ها می‌باشد. بعضی از مادربردها دارای دو عدد، BIOS هستند. که از BIOS دوم، به عنوان پشتیبان، استفاده می‌گردد.

BIOS فقط حاوی دستورالعمل‌ها می‌باشد. مقادیر تنظیمات انجام شده توسط کاربر، مثل تاریخ، رمز عبور و غیره در حافظه‌ای به نام CMOS ذخیره می‌گردد. حافظه‌ی CMOS دارای یک باتری، برای نگهداری محتویات حافظه می‌باشد. در صورتیکه باتری را برداریم و یا باتری ضعیف شده باشد، اطلاعات CMOS پاک شده و BIOS از اطلاعات پیش‌فرض خود استفاده نموده و سیستم را راه‌اندازی می‌نماید.

UEFI

UEFI یا Unified Extensible Firmware Interface نسخه‌ی توسعه یافته از EFI می‌باشد. UEFI به عنوان جایگزینی برای BIOS در نظر گرفته شده است. UEFI دارای قابلیت پشتیبانی از فایل سیستم است. UEFI، از دیسک‌هایی که، با استاندارد GPT، پارتیشن‌بندی شده باشند، پشتیبانی می‌کند. همچنین UEFI، دارای رابط کاربری بهتری می‌باشد.

مرحله‌ی دوم MBR/GPT

MBR

MBR یا Master Boot Record سکتور صفر دیسک و یا سکتور بوت از دیسک است. این بخش در ابتدای هر دیسک قرار می‌گیرد و در آن، اطلاعات مربوط به پارتیشن‌های دیسک و همچنین کدهای اولیه، مربوط به bootloader، در آن، ذخیره می‌گردد. این استاندارد برای پارتیشن‌های تا ظرفیت ۲ ترابایت قابل استفاده می‌باشد. در این روش شما بیش از چهار پارتیشن اصلی نمی‌توانید ایجاد نمایید. برای ایجاد پارتیشن‌های بیشتر شما می‌توانید سه پارتیشن اصلی و پارتیشن‌های دیگر را به صورت extended داشته باشید.

GPT

GPT یا GUID Partition Table به عنوان جایگزین برای MBR ایجاد شده است. با استفاده از GPT می‌توانید، تعداد ۱۲۸ پارتیشن‌های اصلی ایجاد نمایید. این روش سایز پارتیشن‌هایی را که پشتیبانی می‌نماید، ۹.۴ ZB است.

مرحله‌ی سوم Bootloader

در این مرحله، bootloader برای اجرا، بارگذاری می‌شود. bootloader، هسته‌ی سیستم عامل را در حافظه‌ی اصلی بارگذاری کرده و مدیریت سیستم را به آن واگذار می‌نماید. بدون بوت لودر، نمی‌توان سیستم عامل را، بارگذاری نمود. bootloader در لینوکس، دارای انواعی چون Grub و Grub2 و LILO می‌باشد. که در حال حاضر عموماً از GRUB2 استفاده می‌گردد.

مرحله‌ی چهارم Kernel

در ادامه‌ی مراحل بوت شدن لینوکس، هسته‌ی سیستم عامل، توسط bootloader، در حافظه‌ی اصلی بار می‌شود. کرنل یا هسته سیستم عامل، نقش اصلی را، در ارتباط برنامه‌های مختلف با سخت‌افزار و مدیریت آن‌ها، ایفا می‌کند.

کرنل در ابتدا به صورت فشرده بوده و پس از بار شدن در حافظه، خودش را، از فشرده‌گی خارج می‌کند. سپس مدیریت سیستم را بدست می‌گیرد.

مرحله‌ی پنجم init/systemd

اولین process، یا فرآیندی که، کرنل اجرا می‌کند، systemd می‌باشد. systemd، در بیشتر توزیع‌های لینوکس، استفاده می‌گردد. این فرآیند جایگزین گونه‌ی قدیمی فرآیند SysV init، شده است. systemd، والد تمام فرآیندهای سیستم عامل است. وظیفه‌ی systemd مدیریت فرآیندها، می‌باشد.

مرحله‌ی ششم Runlevel/Targets

Runlevels

در این مرحله، در صورتی که، فرآیند اولیه، فرآیند init باشد، از سطوح اجرا (Runlevel) برای مشخص کردن سرویس‌هایی که باید در هر سطح، اجرا شوند، استفاده می‌کنیم. در فرآیند init هفت سطح اجرایی، تعریف گردیده است.

Targets

در صورتی که فرآیند اول ما systemd باشد، سرویس‌هایی که باید اجرا شوند در فایل `/etc/systemd/system/default.target` قرار دارند. به عنوان مثال، برای محیط دسکتاپ، target انتخاب شده، معادل runlevel 5 در سیستم قدیمی init می‌باشد، که حالت چندکاربره‌ی گرافیکی است.

۲.۲ هسته‌ی سیستم عامل / اعمال تغییرات و کامپایل مجدد

ابتدا کد منبع هسته را دریافت کنید. برای این کار با دستور `uname -r` نسخه هسته خود را ببینید. سپس از دستور زیر استفاده کنید:

```
apt-get install linux-source-x.x.x
```

ابزارهای لازم برای کامپایل و نصب هسته را دریافت کنید:

```
apt-get install build-essential fakeroot
```

```
apt-get build-dep linux
```

دقت کنید که لزومی ندارد نسخه کد منبعی که دریافت می‌کنید با نسخه هسته خودتان یکسان باشد. در اینجا صرفاً برای اینکه نسخه بروز را دریافت کنید از نسخه هسته خودتان استفاده کرده‌اید.

به کمک دستور زیر، کدهای هسته را در یک پوشه مشخص بازگشایی کنید:

apt-get source linux

یک پوشه با نام linux-source-x.x.x ایجاد شده که حاوی کد هسته ی لینوکس می باشد.

۲.۳ فعالیت ها

به کمک [لینک](#) (محتوای این صفحه، ضمیمه همین فایل شده است) نحوه کامپایل کردن هسته و نصب آن را به اختصار بیان کنید. سپس هسته ی سیستم عامل را یک بار کامپایل نمایید. در دفعه اول این کار زمان گیر خواهد بود، ولی عملیات را برای دفعات بعد تسریع خواهد کرد.

ضمیمه ۱:

How to Compile a Linux Kernel

By

Jack Wallen

-

April 27, 2018

137186

Once upon a time the idea of upgrading the Linux kernel sent fear through the hearts of many a user. Back then, the process of upgrading the kernel involved a lot of steps and even more time. Now, installing a new kernel can be easily handled with package managers like apt. With the addition of certain repositories, you can even easily install experimental or specific kernels (such as real-time kernels for audio production) without breaking a sweat.

Considering how easy it is to upgrade your kernel, why would you bother compiling one yourself? Here are a few possible reasons:

- You simply want to know how it's done.
- You need to enable or disable specific options into a kernel that simply aren't available via the standard options.
- You want to enable hardware support that might not be found in the standard kernel.
- You're using a distribution that requires you compile the kernel.
- You're a student and this is an assignment.

Regardless of why, knowing how to compile a Linux kernel is very useful and can even be seen as a right of passage. When I first compiled a new Linux kernel (a long, long time ago) and managed to boot from said kernel, I felt a certain thrill coursing through my system (which was quickly crushed the next time I attempted and failed). With that said, let's walk through the process of compiling a Linux kernel. I'll be demonstrating on Ubuntu 16.04 Server. After running through a standard `sudo apt upgrade`, the installed kernel is 4.4.0-121. I want to upgrade to kernel 4.17. Let's take care of that.

A word of warning: I highly recommend you practice this procedure on a virtual machine. By working with a VM, you can always create a snapshot and back out of any problems with ease. DO NOT upgrade the kernel this way on a production machine... not until you know what you're doing.

Downloading the kernel

The first thing to do is download the kernel source file. This can be done by finding the URL of the kernel you want to download (from [Kernel.org](https://kernel.org)). Once you have the URL, download the source file with the following command (I'll demonstrate with kernel 4.17 RC2):

```
wget https://git.kernel.org/torvalds/t/linux-4.17-rc2.tar.gz
```

While that file is downloading, there are a few bits to take care of.

Installing requirements

In order to compile the kernel, we'll need to first install a few requirements. This can be done with a single command:

```
sudo apt-get install git fakeroot build-essential ncurses-dev xz-utils libssl-dev bc flex libelf-dev bison
```

Do note: You will need at least 12GB of free space on your local drive to get through the kernel compilation process. So make sure you have enough space.

Extracting the source

From within the directory housing our newly downloaded kernel, extract the kernel source with the command:

```
tar xvzf linux-4.17-rc2.tar.gz
```

Change into the newly created directory with the command `cd linux-4.17-rc2`.

Configuring the kernel

Before we actually compile the kernel, we must first configure which modules to include. There is actually a really easy way to do this. With a single command, you can copy the current kernel's config file and then use the tried and true `menuconfig` command to make any necessary changes.

To do this, issue the command:

```
cp /boot/config-$(uname -r) .config
```

Now that you have a configuration file, issue the command *make menuconfig*. This command will open up a configuration tool (Figure 1) that allows you to go through every module available and enable or disable what you need or don't need.

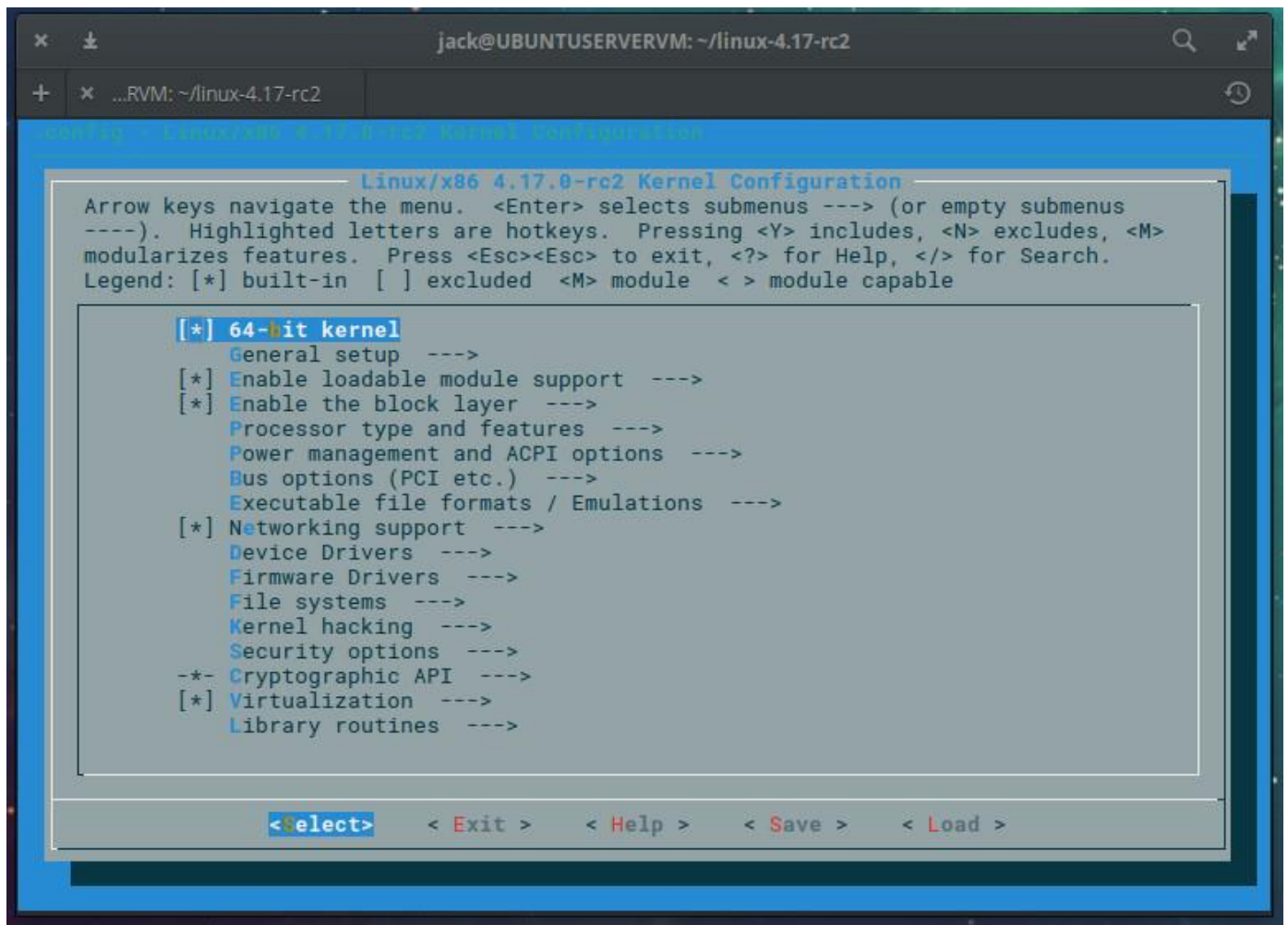
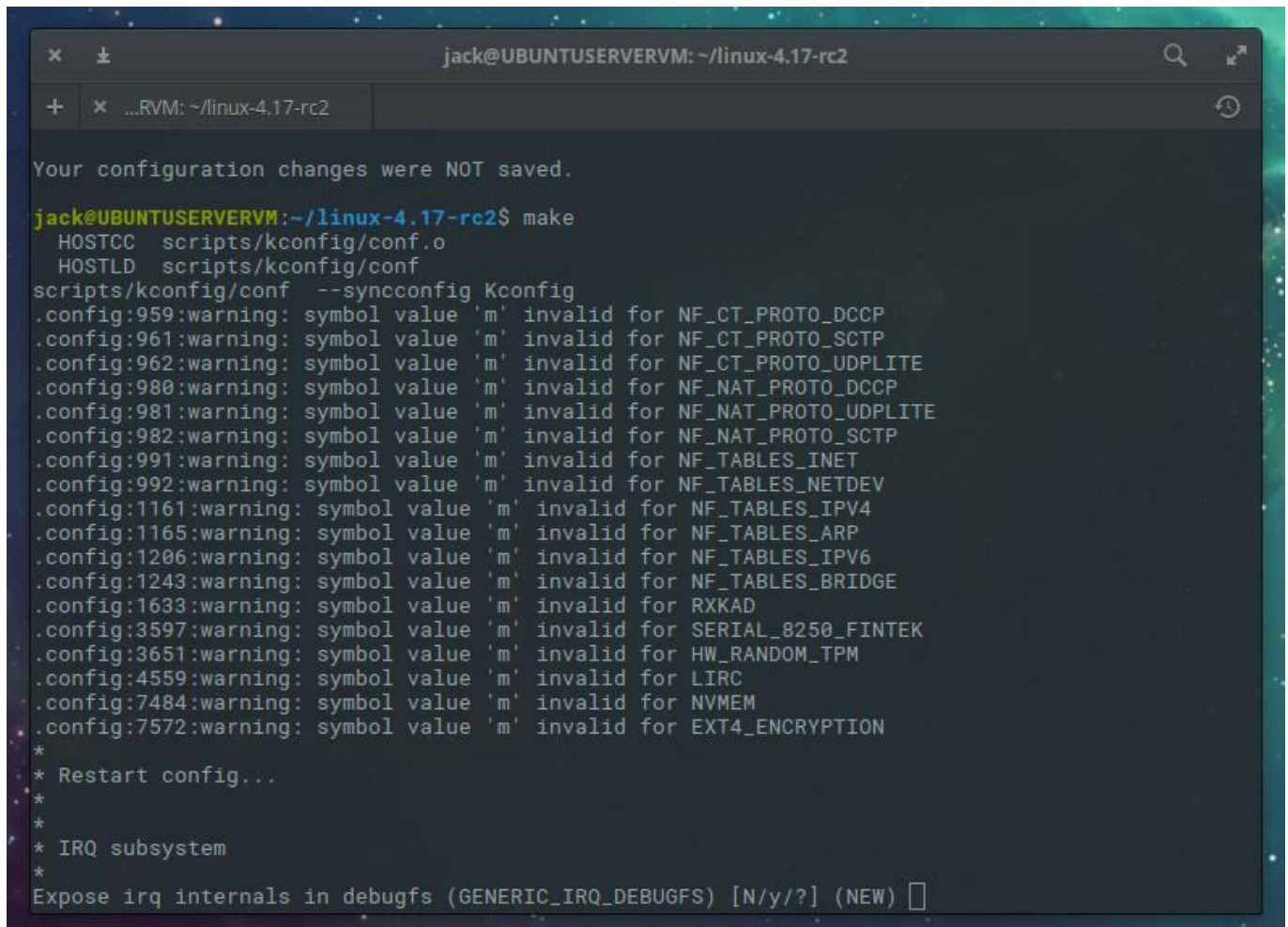


Figure 1: The make menuconfig in action.

It is quite possible you might disable a critical portion of the kernel, so step through menuconfig with care. If you're not sure about an option, leave it alone. Or, better yet, stick with the configuration we just copied from the running kernel (as we know it works). Once you've gone through the entire list (it's quite long), you're ready to compile!

Compiling and installing

Now it's time to actually compile the kernel. The first step is to compile using the make command. So issue make and then answer the necessary questions (Figure 2). The questions asked will be determined by what kernel you're upgrading from and what kernel you're upgrading to. Trust me when I say there's a ton of questions to answer, so give yourself plenty of time here.



```
jack@UBUNTUSERVERVM: ~/linux-4.17-rc2
Your configuration changes were NOT saved.

jack@UBUNTUSERVERVM:~/linux-4.17-rc2$ make
HOSTCC scripts/kconfig/conf.o
HOSTLD scripts/kconfig/conf
scripts/kconfig/conf --syncconfig Kconfig
.config:959:warning: symbol value 'm' invalid for NF_CT_PROTO_DCCP
.config:961:warning: symbol value 'm' invalid for NF_CT_PROTO_SCTP
.config:962:warning: symbol value 'm' invalid for NF_CT_PROTO_UDPLITE
.config:980:warning: symbol value 'm' invalid for NF_NAT_PROTO_DCCP
.config:981:warning: symbol value 'm' invalid for NF_NAT_PROTO_UDPLITE
.config:982:warning: symbol value 'm' invalid for NF_NAT_PROTO_SCTP
.config:991:warning: symbol value 'm' invalid for NF_TABLES_INET
.config:992:warning: symbol value 'm' invalid for NF_TABLES_NETDEV
.config:1161:warning: symbol value 'm' invalid for NF_TABLES_IPV4
.config:1165:warning: symbol value 'm' invalid for NF_TABLES_ARP
.config:1206:warning: symbol value 'm' invalid for NF_TABLES_IPV6
.config:1243:warning: symbol value 'm' invalid for NF_TABLES_BRIDGE
.config:1633:warning: symbol value 'm' invalid for RXKAD
.config:3597:warning: symbol value 'm' invalid for SERIAL_8250_FINTEK
.config:3651:warning: symbol value 'm' invalid for HW_RANDOM_TPM
.config:4559:warning: symbol value 'm' invalid for LIRC
.config:7484:warning: symbol value 'm' invalid for NVMEM
.config:7572:warning: symbol value 'm' invalid for EXT4_ENCRYPTION
*
* Restart config...
*
*
* IRQ subsystem
*
Expose irq internals in debugfs (GENERIC_IRQ_DEBUGFS) [N/y/?] (NEW) □
```

Figure 2: Answering the questions for the make command.

After answering the litany of questions, you can then install the modules you've enabled with the command:

```
make modules_install
```

Once again, this command will take some time, so either sit back and watch the output, or go do something else (as it will not require your input). Chances are, you'll want to undertake another task (unless you really enjoy watching output fly by in a terminal).

Now we install the kernel with the command:

```
sudo make install
```

Again, another command that's going to take a significant amount of time. In fact, the `make install` command will take even longer than the `make modules_install` command. Go have lunch, configure a router, install Linux on a few servers, or take a nap.

Enable the kernel for boot

Once the `make install` command completes, it's time to enable the kernel for boot. To do this, issue the command:

```
sudo update-initramfs -c -k 4.17-rc2
```

Of course, you would substitute the kernel number above for the kernel you've compiled. When that command completes, update grub with the command:

```
sudo update-grub
```

You should now be able to restart your system and select the newly installed kernel.

Congratulations!

You've compiled a Linux kernel! It's a process that may take some time; but, in the end, you'll have a custom kernel for your Linux distribution, as well as an important skill that many Linux admins tend to overlook.