The kernel report

(Kernel Recipes 2016 edition)

Jonathan Corbet
LWN.net
corbet@lwn.net

Recent releases

Version	Date	Days	Devs	Changesets
4.3	Nov 1	63	1,625	12,274
4.4	Jan 10	70	1,575	13,071
4.5	Mar 13	63	1,537	12,080
4.6	May 15	63	1,678	13,517
4.7	Jul 17	70	1,582	12,283

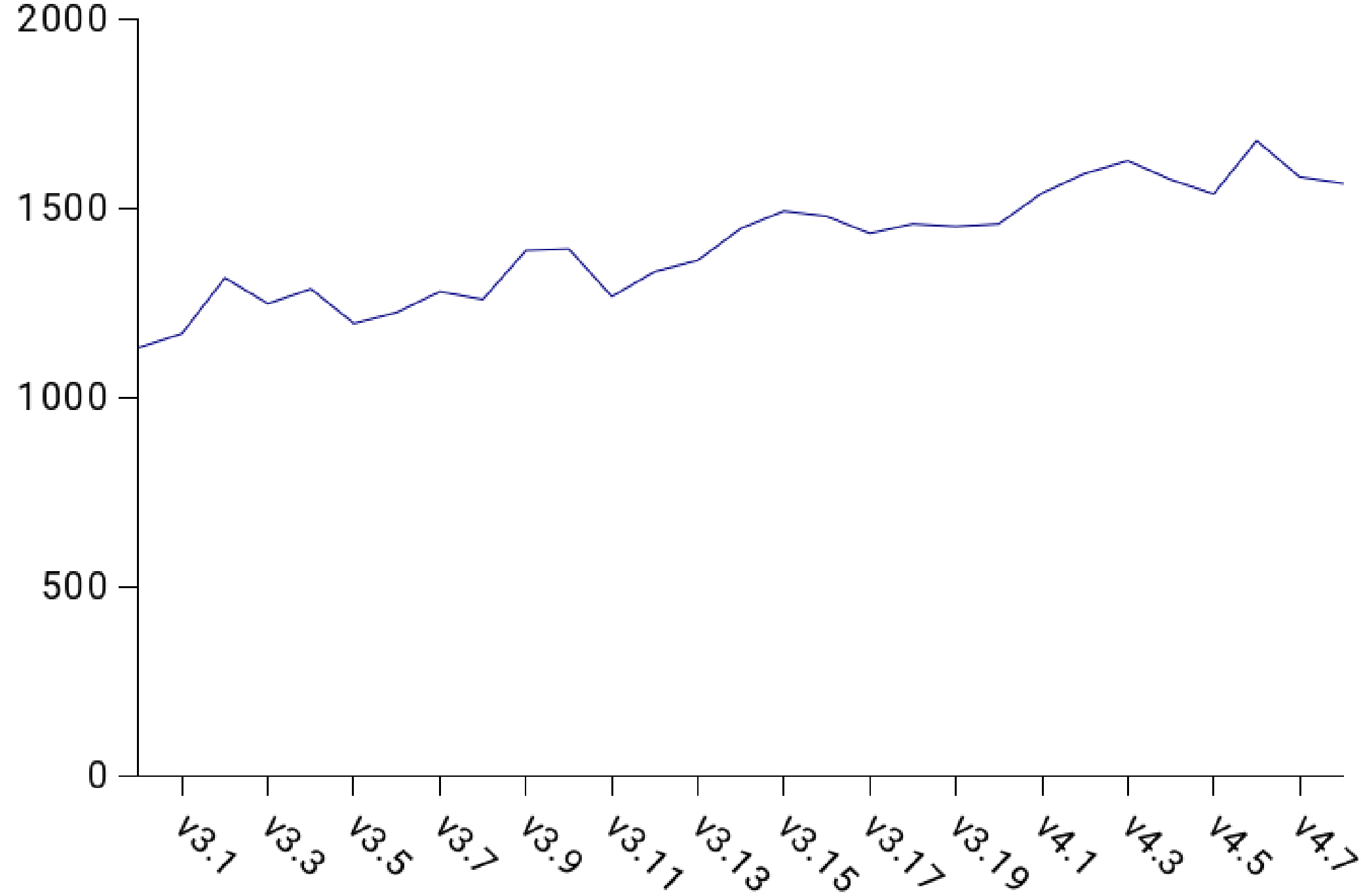
Recent releases

Version	Date	Days	Devs	Changesets
4.3	Nov 1	63	1,625	12,274
4.4	Jan 10	70	1,575	13,071
4.5	Mar 13	63	1,537	12,080
4.6	May 15	63	1,678	13,517
4.7	Jul 17	70	1,582	12,283
4.8	Oct 2	70	1,578*	13,253

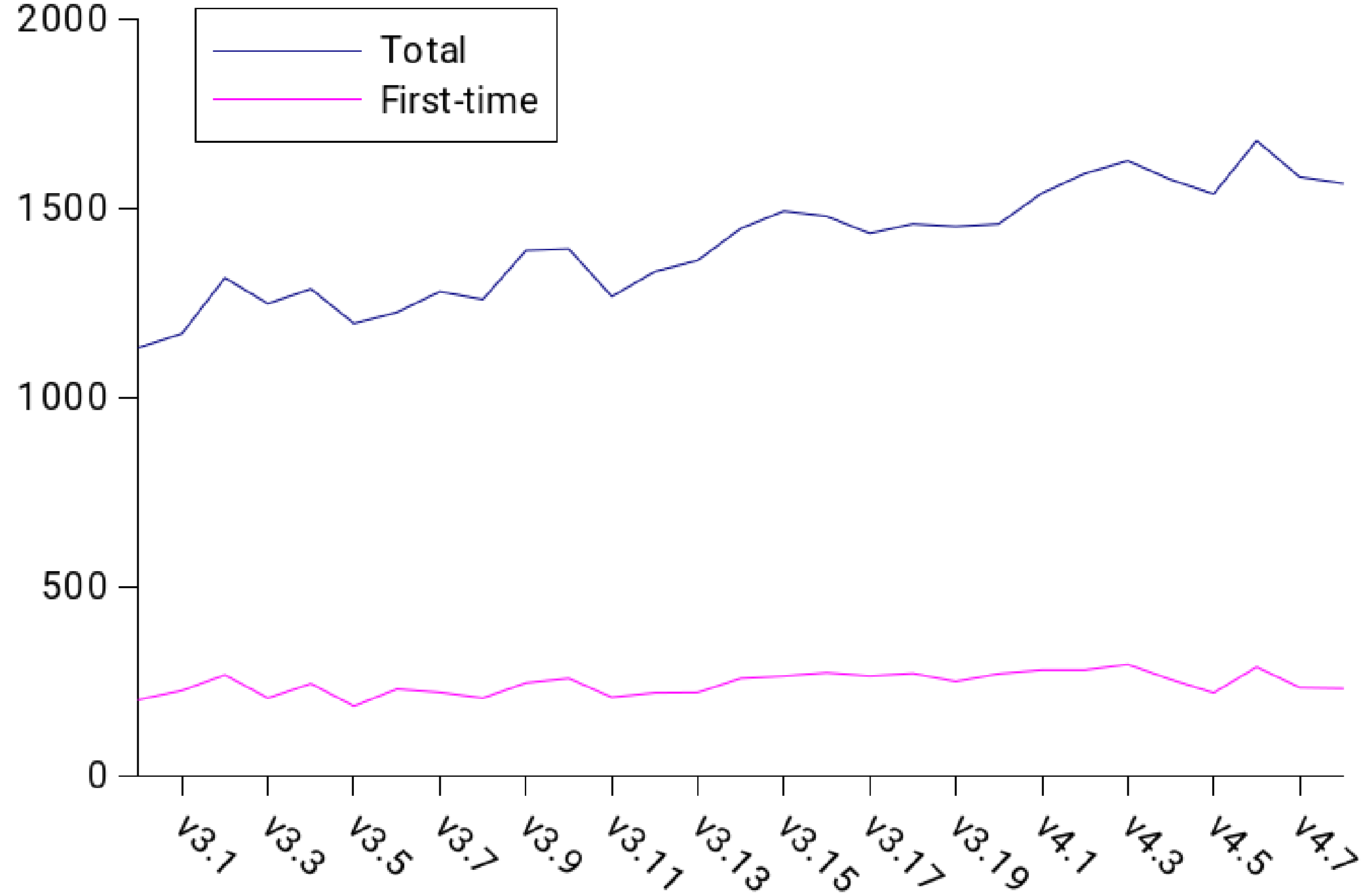
Recent releases

Version	Date	Days	Devs	Changesets
4.3	Nov 1	63	1,625	12,274
4.4	Jan 10	70	1,575	13,071
4.5	Mar 13	63	1,537	12,080
4.6	May 15	63	1,678	13,517
4.7	Jul 17	70	1,582	12,283
4.8	Oct 2	70	1,578*	13,253
Total since 4.2		395	4,028	76,478

Developers contributing to each release



Developers contributing to each release



What’s changing in the develoment community?

Not much!

The process continues to run smoothly


Security

The year in CVE numbers

CVE-2016-0723 CVE-2016-0728 CVE-2016-0758 CVE-2016-0774 CVE-2016-0821
CVE-2016-0823 CVE-2016-1237 CVE-2016-1575 CVE-2016-1576 CVE-2016-1583
CVE-2016-2053 CVE-2016-2059 CVE-2016-2061 CVE-2016-2062 CVE-2016-2063
CVE-2016-2064 CVE-2016-2065 CVE-2016-2066 CVE-2016-2067 CVE-2016-2068
CVE-2016-2069 CVE-2016-2070 CVE-2016-2085 CVE-2016-2117 CVE-2016-2143
CVE-2016-2184 CVE-2016-2185 CVE-2016-2186 CVE-2016-2187 CVE-2016-2188
CVE-2016-2383 CVE-2016-2384 CVE-2016-2543 CVE-2016-2544 CVE-2016-2545
CVE-2016-2546 CVE-2016-2547 CVE-2016-2548 CVE-2016-2549 CVE-2016-2550
CVE-2016-2782 CVE-2016-2847 CVE-2016-2853 CVE-2016-2854 CVE-2016-3070
CVE-2016-3134 CVE-2016-3135 CVE-2016-3136 CVE-2016-3137 CVE-2016-3138
CVE-2016-3139 CVE-2016-3140 CVE-2016-3156 CVE-2016-3157 CVE-2016-3672
CVE-2016-3689 CVE-2016-3707 CVE-2016-3713 CVE-2016-3841 CVE-2016-3951
CVE-2016-3955 CVE-2016-3961 CVE-2016-4440 CVE-2016-4470 CVE-2016-4482
CVE-2016-4485 CVE-2016-4486 CVE-2016-4557 CVE-2016-4558 CVE-2016-4565
CVE-2016-4568 CVE-2016-4569 CVE-2016-4578 CVE-2016-4580 CVE-2016-4581
CVE-2016-4794 CVE-2016-4805 CVE-2016-4913 CVE-2016-4951 CVE-2016-4997
CVE-2016-4998 CVE-2016-5243 CVE-2016-5244 CVE-2016-5340 CVE-2016-5342
CVE-2016-5344 CVE-2016-5400 CVE-2016-5412 CVE-2016-5696 CVE-2016-5728
CVE-2016-5828 CVE-2016-5829 CVE-2016-6130 CVE-2016-6136 CVE-2016-6156
CVE-2016-6162 CVE-2016-6187 CVE-2016-6197 CVE-2016-6198 CVE-2016-6480
CVE-2016-6516

Our security algorithm

When a vulnerability is found:

- Create a patch with a fix
- Distributors ship an update

This approach has some problems...

Photo: Jeffrey Kontur

Our security algorithm

When a vulnerability is found:

- Create a patch with a fix
- Distributors ship an update

I have an example of a security bug that a Google researcher found in a 3.10 kernel (but not mainline) I fixed and pushed out an update, but never got picked up in Nexus phones until 6 months later when I found the right person/group to poke within Google.

That was a 6 month window where anyone could have gotten root on your phone, easily.
— Greg Kroah-Hartman

Our security algorithm

When a vulnerability is found:

- Create a patch with a fix
- ~~Distributors ship an update~~


LWN


LWN

Our defenses are not sufficient for today’s threats

Vulnerabilities will always be with us

Vulnerabilities will always be with us

We need to be eliminating *classes of exploits*

Eliminating classes of exploits

Post-init read-only memory (v4.6)

Eliminating classes of exploits

Post-init read-only memory (v4.6)
Use of GCC plugins (v4.8)

Eliminating classes of exploits

Post-init read-only memory (v4.6)
Use of GCC plugins (v4.8)
Kernel stack hardening (v4.9?)

Eliminating classes of exploits

Post-init read-only memory (v4.6)
Use of GCC plugins (v4.8)
Kernel stack hardening (v4.9?)
Hardened user-copy (v4.8)

Eliminating classes of exploits

Post-init read-only memory (v4.6)
Use of GCC plugins (v4.8)
Kernel stack hardening (v4.9?)
Hardened user-copy (v4.8)
Reference-count hardening

Eliminating classes of exploits

Post-init read-only memory (v4.6)
Use of GCC plugins (v4.8)
Kernel stack hardening (v4.9?)
Hardened user-copy (v4.8)
Reference-count hardening

Much of this originates in grsecurity.net
Some funded by CII

What’s the catch?

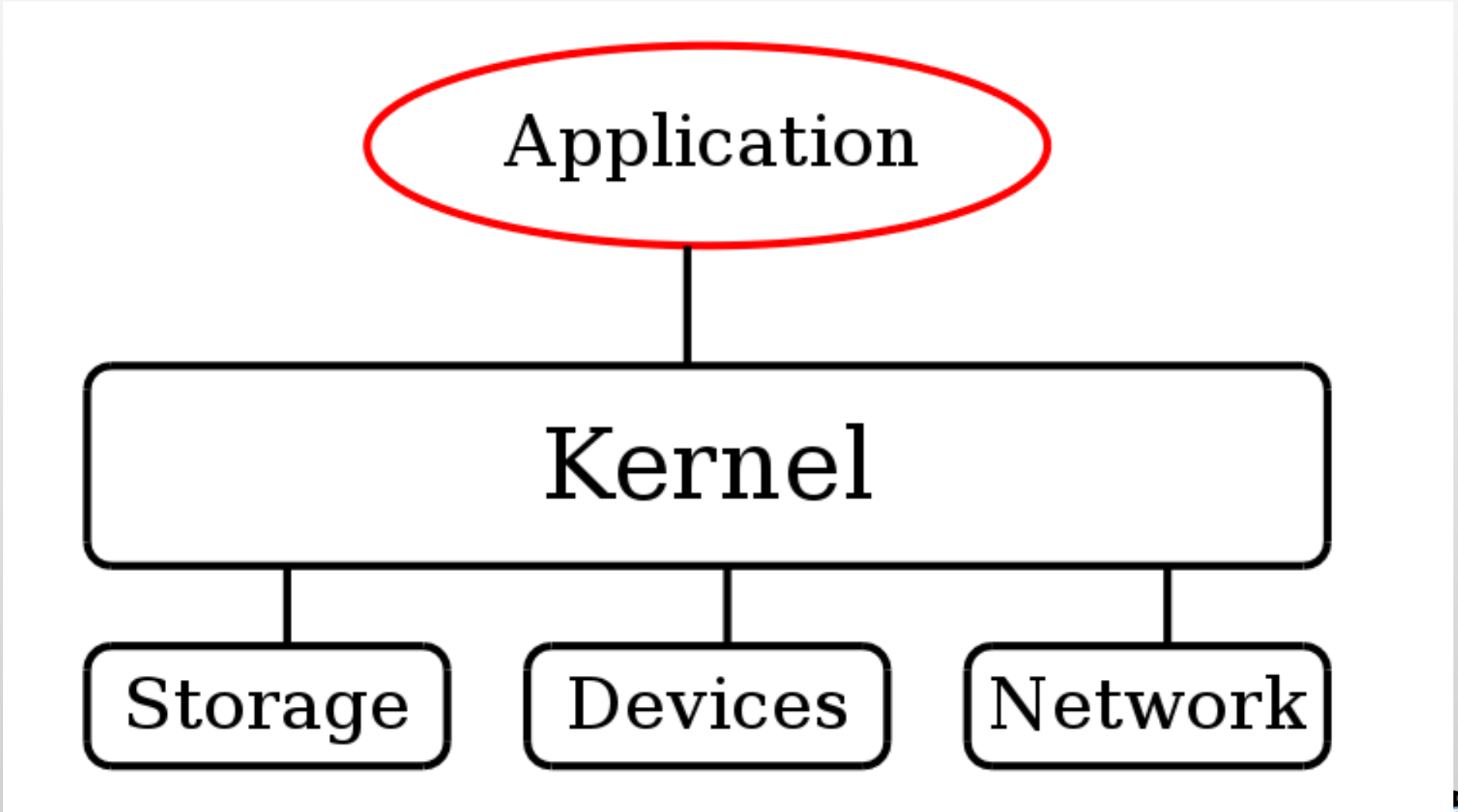
Security-related code has tradeoffs:

- Performance costs
- User-space compatibility issues

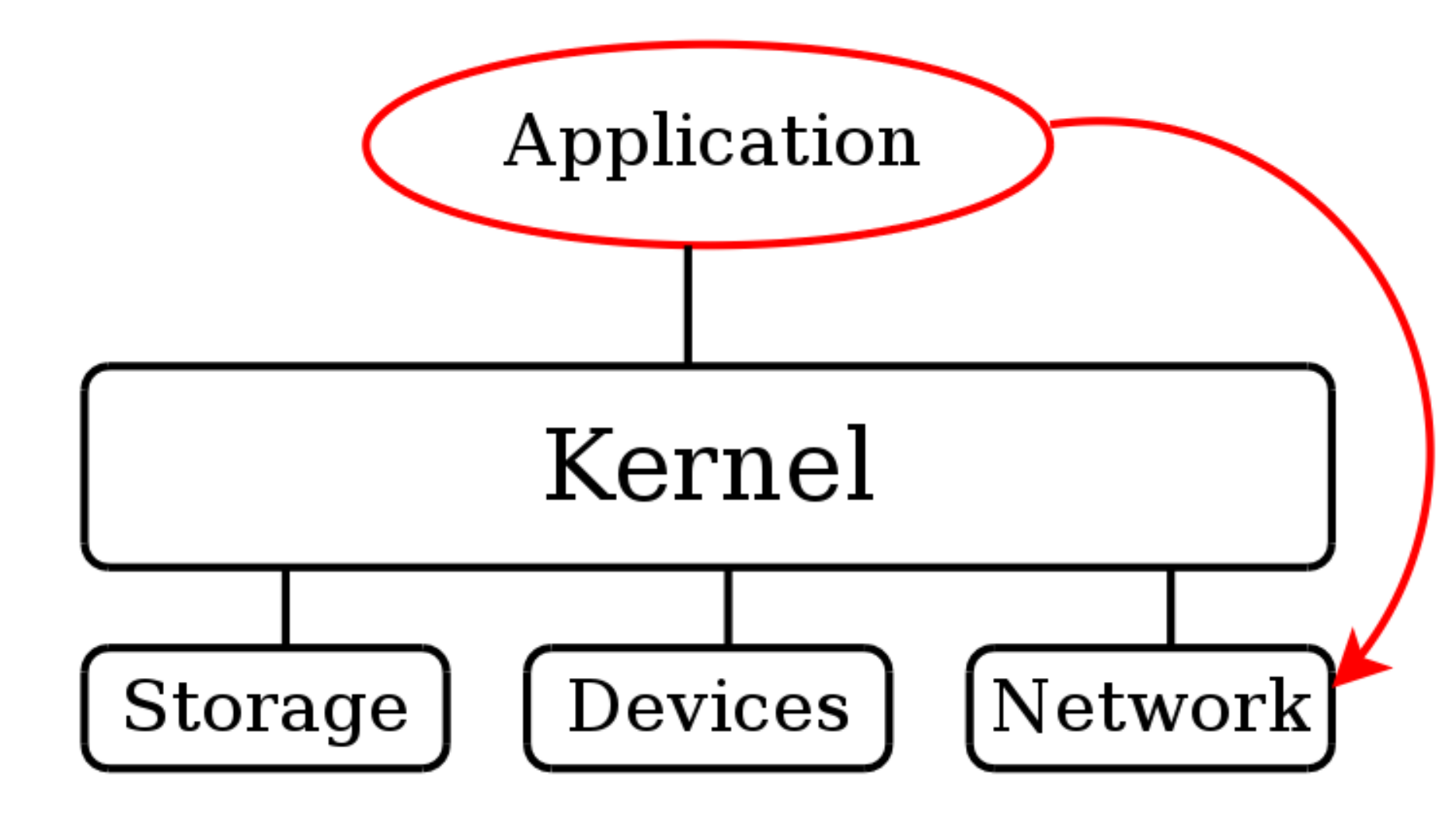
Can we convince developers it’s worth the cost?

Bypassing the kernel

Kernel bypass



Kernel bypass



Transport over UDP (TOU)

Use UDP to move packets around the net

Embed higher-level protocols within UDP packets

Transport protocols can be done in user space!

(See also: QUIC)

Why TOU?

Faster deployment of protocol enhancements

Why TOU?

Faster deployment of protocol enhancements

> “The TCP stack in the Android/iOS/Windows kernel is so out of date that in order to get even moderately recent TCP features it is necessary to do this.”
> — David Miller, June 2016

Why TOU?

Faster deployment of protocol enhancements

Avoid middlebox interference

Why TOU?

Faster deployment of protocol enhancements

Avoid middlebox interference

- Protocol deployment
- End-to-end privacy

Why **not** TOU?

About those protocol enhancements

- They don’t have to be free
- They don’t have to become part of the public net

Do we want every app to speak its own protocol?

Will the kernel still be a strong unifying force for the net?

BPF


Choose Your
BPF
(Best Princess Friend)
and Share your Dreams Together!
Cinderella
Aurora
Ariel
Tiana
Belle
Not Sure?
Take a QUIZ!

The Berkeley Packet Filter

A simple in-kernel virtual machine

Users can load executable code into the kernel
- bpf() system call

Data can be exchanged with the kernel or user space

This sounds dangerous...?

Lots of rules for BPF programs

- No loops
- No access to arbitrary memory
- No leaking kernel pointers to user space
- No access to uninitialized data
- Blinding of constants in programs
- ...

Limited to root in many cases

- but not all

Uses of BPF

Filtering of packets to a socket
System call restriction via seccomp()
Perf events filtering
Packet classification and queuing
Tracepoint data filtering and analysis
Early device-level packet filter/drop/forward (XDP)

Uses of BPF

Filtering of packets to a socket
System call restriction via seccomp()
Perf events filtering
Packet classification and queuing
Tracepoint data filtering and analysis
Early device-level packet filter/drop/forward (XDP)
…?

Our brave new BPF world

Chunks of important kernel code come from user space.

Stable kernels and backports

Remember 2.4?

Multiple years between releases

Huge feature gaps to fill

Distributors backported lots of code

...and shipped out-of-tree features

What did we do about it?

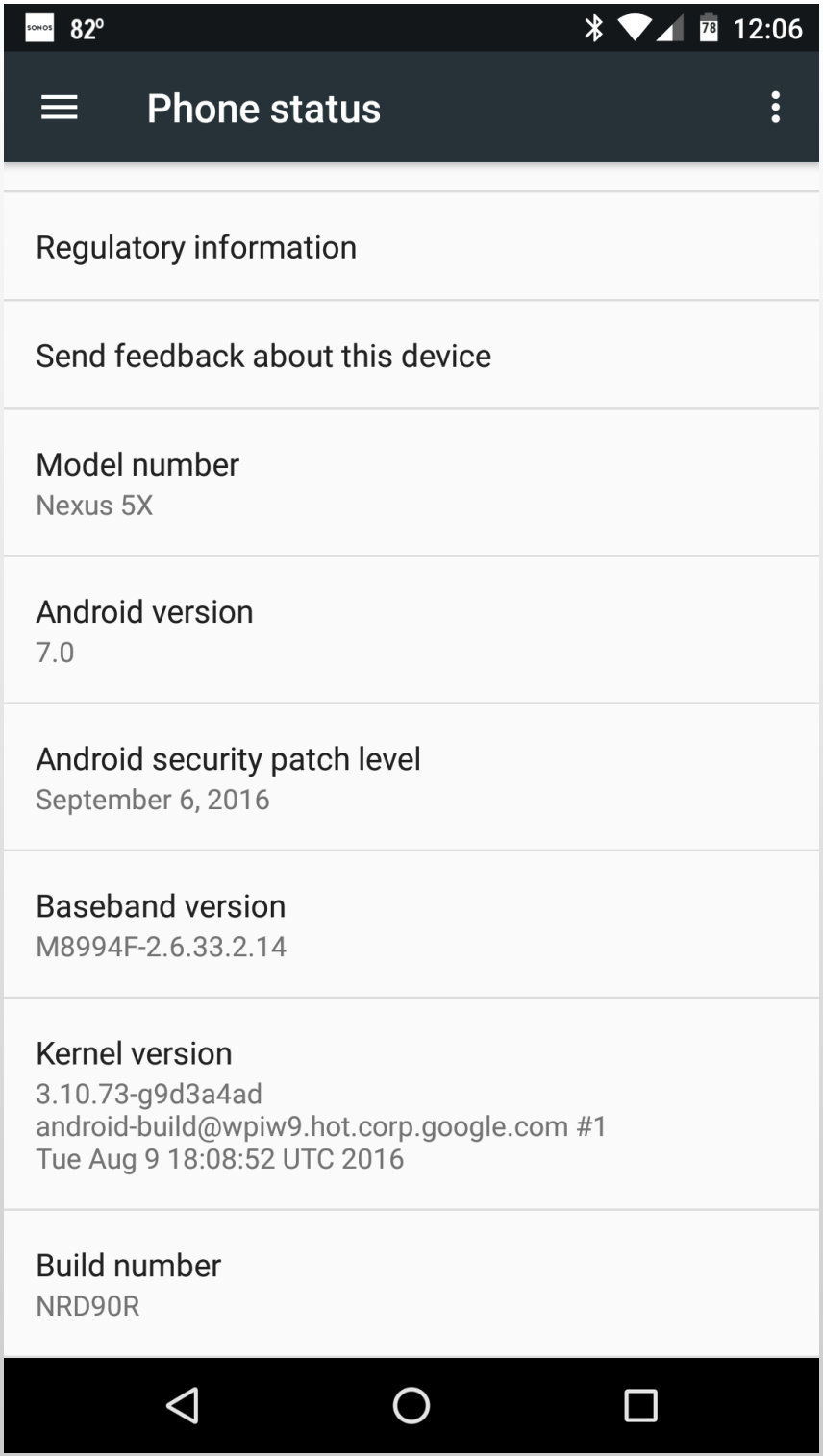
The “upstream first” rule

What did we do about it?

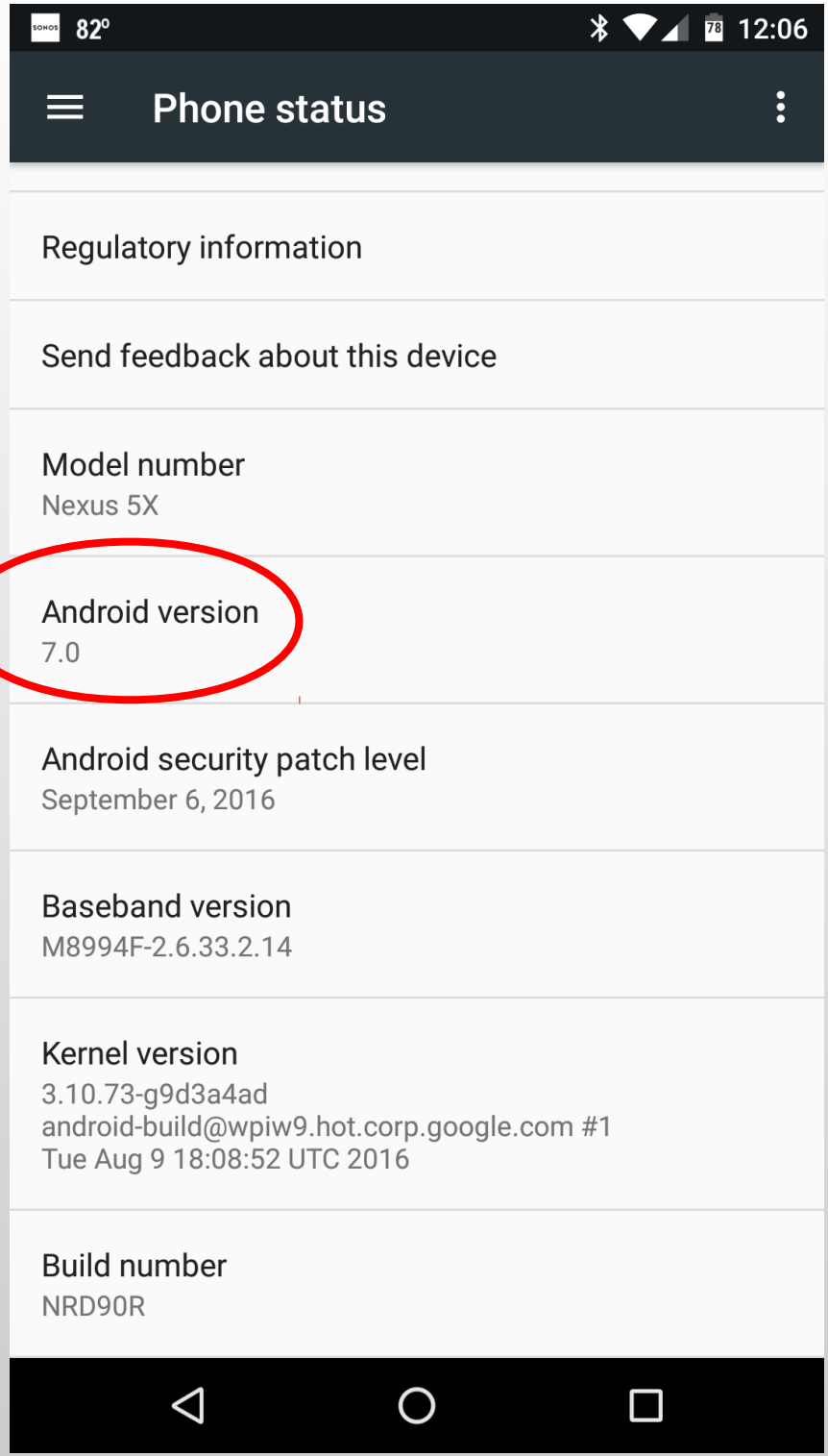
The “upstream first” rule

The “new” development model

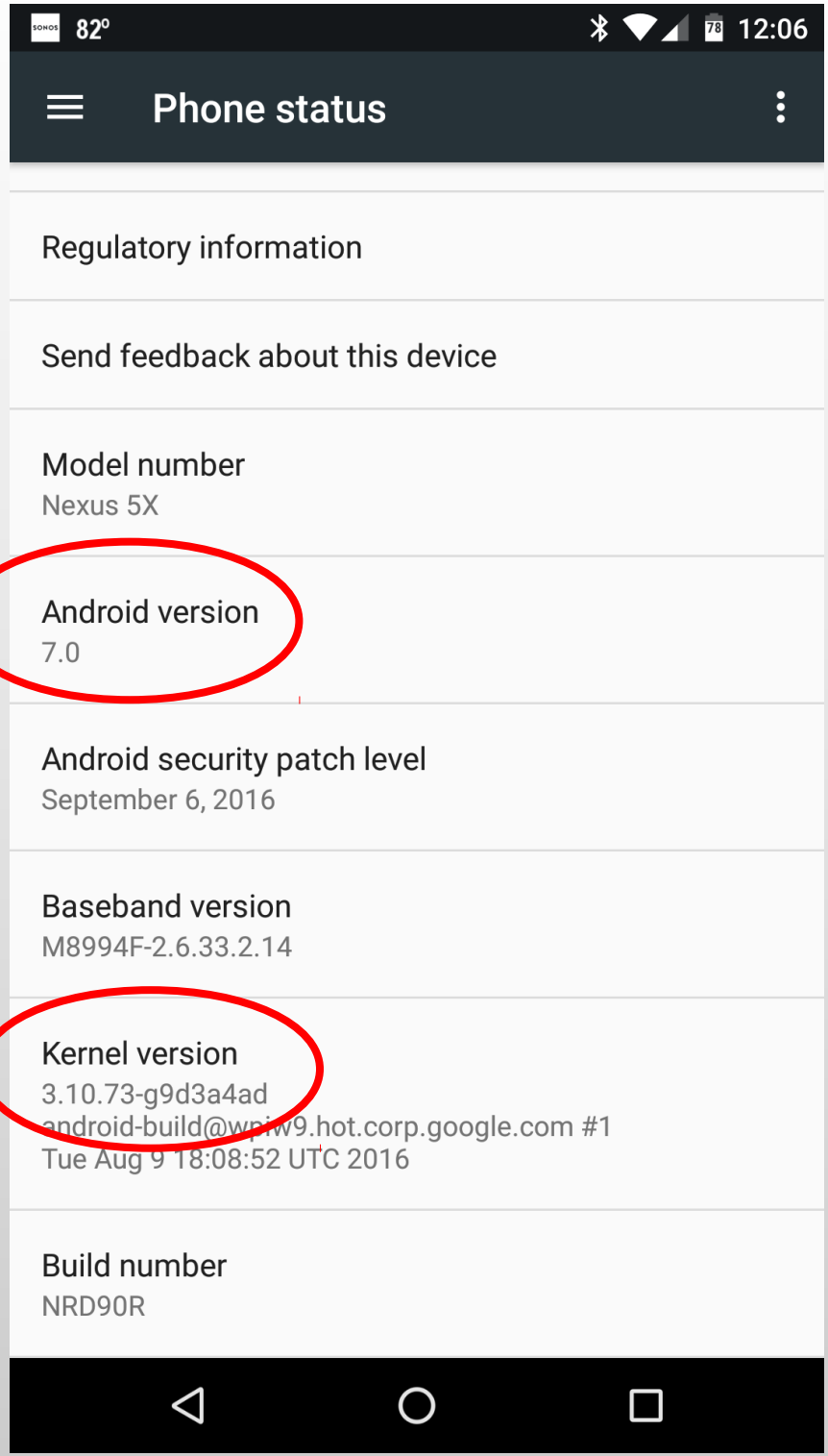
Problem solved!


82°
12:06
Phone status
Regulatory information
Send feedback about this device
Model number
Nexus 5X
Android version
7.0
Android security patch level
September 6, 2016
Baseband version
M8994F-2.6.33.2.14
Kernel version
3.10.73-g9d3a4ad
android-build@wpiw9.hot.corp.google.com #1
Tue Aug 9 18:08:52 UTC 2016
Build number
NRD90R


LWN


82°
12:06
Phone status
Regulatory information
Send feedback about this device
Model number
Nexus 5X
Android version
7.0
Android security patch level
September 6, 2016
Baseband version
M8994F-2.6.33.2.14
Kernel version
3.10.73-g9d3a4ad
android-build@wpiw9.hot.corp.google.com #1
Tue Aug 9 18:08:52 UTC 2016
Build number
NRD90R


LWN


82°
12:06
Phone status
Regulatory information
Send feedback about this device
Model number
Nexus 5X
Android version
7.0
Android security patch level
September 6, 2016
Baseband version
M8994F-2.6.33.2.14
Kernel version
3.10.73-g9d3a4ad
android-build@wpiw9.hot.corp.google.com #1
Tue Aug 9 18:08:52 UTC 2016
Build number
NRD90R


LWN

The 3.10 kernel

Was released in June 2013

- 3.10.73 update was March 2015

Is 221,430 patches behind the mainline

Fear of mainline kernels

The possibility of new bugs and regressions

Mobile SoC code out-of-tree

Company	Phone	SOC	Insertions
LG	G3	Msm	2.6 M
Motorola	Moto X	Msm	1.8 M
Samsung	Galaxy 4	Exynos	1.1 M
Samsung	Galaxy S5	Msm	3.1 M
Sony	Xperia Z2	Msm	1.8 M
Sony	Xperia C	Mediatek	1.9 M
Acer	Liquid E2	Mediatek	1.4 M
Asus	Zenfone 6	Atom	2.2 M
Huawei	Ascend P7	Hisilicon	2.7 M

Fear of mainline kernels

The possibility of new bugs and regressions

Vast amount of out-of-tree code to forward port

Why all that out-of-tree code?

Upstreaming can take a long time
- Wakelocks
- USB charging

Why all that out-of-tree code?

Upstreaming can take a long time
- Wakelocks
- USB charging

Some of it is not upstreamable
- Scheduler rewrites

Why all that out-of-tree code?

Upstreaming can take a long time
- Wakelocks
- USB charging

Some of it is not upstreamable
- Scheduler rewrites

The kernel moves too slowly!

Two points of view

Kernel developers

“We’ve been doing this for 25 years and plan to still be here 25 years from now.”

Two points of view

Kernel developers

“We’ve been doing this for 25 years and plan to still be here 25 years from now.”

CE manufacturers

“Nobody will remember this product next year.”

Perhaps our biggest process problem at the moment

Copyleft

The problem

Lots of companies shipping the kernel without complying with the terms of the GPL.

To sue or not to sue?

Some say:

- Companies will not comply without the threat of consequences

- Lawsuits have yielded useful code contributions in the past

- Without enforcement, the kernel is effectively BSD licensed.

Others respond:

- Lawsuits turn companies and their employees into enemies
- The outcome of legal action is always uncertain
- Little useful code has come that way
- It is better to work with engineers and change companies from within
- We have had great success without lawsuits

Corporate support 4.2..

Intel	10,933	14.3%
unknown	5,682	7.4%
Red Hat	5,625	7.4%
none	4,643	6.1%
Linaro	3,544	4.6%
Samsung	3.089	4.0%
IBM	2,337	3.1%
SUSE	2,123	2.8%
AMD	1,629	2.1%
Renesas Electronics	1,514	2.0%
consultants	1,456	1.9%
Google	1,428	1.9%

Another way to look at it

1) Code for existing devices eventually

— or —

2) Support from companies indefinitely?

How do we best ensure the success of Linux and free software?

Thank you!