**Linux Kernel Release Model**

FEB 5TH, 2018

**Note**

This post is based on a whitepaper I wrote at the beginning of 2016 to be used to help many different companies understand the Linux kernel release model and encourage them to start taking the LTS stable updates more often. I then used it as a basis of a presentation I gave at the [Linux Recipes conference in September 2017](https://kernel-recipes.org/en/2017/) which can be seen [here](https://www.youtube.com/watch?v=RKadXpQLmPU).

With the [recent craziness of Meltdown and Spectre](http://www.kroah.com/log/blog/2018/01/06/meltdown-status/) , I’ve seen lots of things written about how Linux is released and how we handle handles security patches that are totally incorrect, so I figured it is time to dust off the text, update it in a few places, and publish this here for everyone to benefit from.

I would like to thank the reviewers who helped shape the original whitepaper, which has helped many companies understand that they need to stop “cherry picking” random patches into their device kernels. Without their help, this post would be a total mess. All problems and mistakes in here are, of course, all mine. If you notice any, or have any questions about this, please let me know.

**Overview**

This post describes how the Linux kernel development model works, what a long term supported kernel is, how the kernel developers approach security bugs, and why all systems that use Linux should be using all of the stable releases and not attempting to pick and choose random patches.

**Linux Kernel development model**

The Linux kernel is the largest collaborative software project ever. In 2017, over 4,300 different developers from over 530 different companies contributed to the project. There were 5 different releases in 2017, with each release containing between 12,000 and 14,500 different changes. On average, 8.5 changes are accepted into the Linux kernel every hour, every hour of the day. A non-scientific study (i.e. Greg’s mailbox) shows that each change needs to be submitted 2-3 times before it is accepted into the kernel source tree due to the rigorous review and testing process that all kernel changes are put through, so the engineering effort happening is much larger than the 8 changes per hour.

At the end of 2017 the size of the Linux kernel was just over 61 thousand files consisting of 25 million lines of code, build scripts, and documentation (kernel release 4.14). The Linux kernel contains the code for all of the different chip architectures and hardware drivers that it supports. Because of this, an individual system only runs a fraction of the whole codebase. An average laptop uses around 2 million lines of kernel code from 5 thousand files to function properly, while the Pixel phone uses 3.2 million lines of kernel code from 6 thousand files due to the increased complexity of a SoC.

**Kernel release model**

With the release of the 2.6 kernel in December of 2003, the kernel developer community switched from the previous model of having a separate development and stable kernel branch, and moved to a “stable only” branch model. A new release happened every 2 to 3 months, and that release was declared “stable” and recommended for all users to run. This change in development model was due to the very long release cycle prior to the 2.6 kernel (almost 3 years), and the struggle to maintain two different branches of the codebase at the same time.

The numbering of the kernel releases started out being 2.6.x, where x was an incrementing number that changed on every release The value of the number has no meaning, other than it is newer than the previous kernel release. In July 2011, Linus Torvalds changed the version number to 3.x after the 2.6.39 kernel was released. This was done because the higher numbers were starting to cause confusion among users, and because Greg Kroah-Hartman, the stable kernel maintainer, was getting tired of the large numbers and bribed Linus with a fine bottle of Japanese whisky.

The change to the 3.x numbering series did not mean anything other than a change of the major release number, and this happened again in April 2015 with the movement from the 3.19 release to the 4.0 release number. It is not remembered if any whisky exchanged hands when this happened. At the current kernel release rate, the number will change to 5.x sometime in 2018.

**Stable kernel releases**

The Linux kernel stable release model started in 2005, when the existing development model of the kernel (a new release every 2-3 months) was determined to not be meeting the needs of most users. Users wanted bugfixes that were made during those 2-3 months, and the Linux distributions were getting tired of trying to keep their kernels up to date without any feedback from the kernel community. Trying to keep individual kernels secure and with the latest bugfixes was a large and confusing effort by lots of different individuals.

Because of this, the stable kernel releases were started. These releases are based directly on Linus’s releases, and are released every week or so, depending on various external factors (time of year, available patches, maintainer workload, etc.)

The numbering of the stable releases starts with the number of the kernel release, and an additional number is added to the end of it.

For example, the 4.9 kernel is released by Linus, and then the stable kernel releases based on this kernel are numbered 4.9.1, 4.9.2, 4.9.3, and so on. This sequence is usually shortened with the number “4.9.y” when referring to a stable kernel release tree. Each stable kernel release tree is maintained by a single kernel developer, who is responsible for picking the needed patches for the release, and doing the review/release process. Where these changes are found is described below.

Stable kernels are maintained for as long as the current development cycle is happening. After Linus releases a new kernel, the previous stable kernel release tree is stopped and users must move to the newer released kernel.

**Long-Term Stable kernels**

After a year of this new stable release process, it was determined that many different users of Linux wanted a kernel to be supported for longer than just a few months. Because of this, the Long Term Supported (LTS) kernel release came about. The first LTS kernel was 2.6.16, released in 2006. Since then, a new LTS kernel has been picked once a year. That kernel will be maintained by the kernel community for at least 2 years. See the next section for how a kernel is chosen to be a LTS release.

Currently the LTS kernels are the 4.4.y, 4.9.y, and 4.14.y releases, and a new kernel is released on average, once a week. Along with these three kernel releases, a few older kernels are still being maintained by some kernel developers at a slower release cycle due to the needs of some users and distributions.

Information about all long-term stable kernels, who is in charge of them, and how long they will be maintained, can be found on the [kernel.org release page](https://www.kernel.org/category/releases.html).

LTS kernel releases average 9-10 patches accepted per day, while the normal stable kernel releases contain 10-15 patches per day. The number of patches fluctuates per release given the current time of the corresponding development kernel release, and other external variables. The older a LTS kernel is, the less patches are applicable to it, because many recent bugfixes are not relevant to older kernels. However, the older a kernel is, the harder it is to backport the changes that are needed to be applied, due to the changes in the codebase. So while there might be a lower number of overall patches being applied, the effort involved in maintaining a LTS kernel is greater than maintaining the normal stable kernel.

**Choosing the LTS kernel**

The method of picking which kernel the LTS release will be, and who will maintain it, has changed over the years from an semi-random method, to something that is hopefully more reliable.

Originally it was merely based on what kernel the stable maintainer’s employer was using for their product (2.6.16.y and 2.6.27.y) in order to make the effort of maintaining that kernel easier. Other distribution maintainers saw the benefit of this model and got together and colluded to get their companies to all release a product based on the same kernel version without realizing it (2.6.32.y). After that was very successful, and allowed developers to share work across companies, those companies decided to not do that anymore, so future LTS kernels were picked on an individual distribution’s needs and maintained by different developers (3.0.y, 3.2.y, 3.12.y, 3.16.y, and 3.18.y) creating more work and confusion for everyone involved.

This ad-hoc method of catering to only specific Linux distributions was not beneficial to the millions of devices that used Linux in an embedded system and were not based on a traditional Linux distribution. Because of this, Greg Kroah-Hartman decided that the choice of the LTS kernel needed to change to a method in which companies can plan on using the LTS kernel in their products. The rule became “one kernel will be picked each year, and will be maintained for two years.” With that rule, the 3.4.y, 3.10.y, and 3.14.y kernels were picked.

Due to a large number of different LTS kernels being released all in the same year, causing lots of confusion for vendors and users, the rule of no new LTS kernels being based on an individual distribution’s needs was created. This was agreed upon at the annual Linux kernel summit and started with the 4.1.y LTS choice.

During this process, the LTS kernel would only be announced after the release happened, making it hard for companies to plan ahead of time what to use in their new product, causing lots of guessing and misinformation to be spread around. This was done on purpose as previously, when companies and kernel developers knew ahead of time what the next LTS kernel was going to be, they relaxed their normal stringent review process and allowed lots of untested code to be merged (2.6.32.y). The fallout of that mess took many months to unwind and stabilize the kernel to a proper level.

The kernel community discussed this issue at its annual meeting and decided to mark the 4.4.y kernel as a LTS kernel release, much to the surprise of everyone involved, with the goal that the next LTS kernel would be planned ahead of time to be based on the last kernel release of 2016 in order to provide enough time for companies to release products based on it in the next holiday season (2017). This is how the 4.9.y and 4.14.y kernels were picked as the LTS kernel releases.

This process seems to have worked out well, without many problems being reported against the 4.9.y tree, despite it containing over 16,000 changes, making it the largest kernel to ever be released.

Future LTS kernels should be planned based on this release cycle (the last kernel of the year). This should allow SoC vendors to plan ahead on their development cycle to not release new chipsets based on older, and soon to be obsolete, LTS kernel versions.

**Stable kernel patch rules**

The rules for what can be added to a stable kernel release have remained almost identical for the past 12 years. The full list of the rules for patches to be accepted into a stable kernel release can be found in the [Documentation/process/stable\_kernel\_rules.rst](https://www.kernel.org/doc/html/latest/process/stable-kernel-rules.html)kernel file and are summarized here. A stable kernel change:

* must be obviously correct and tested.
* must not be bigger than 100 lines.
* must fix only one thing.
* must fix something that has been reported to be an issue.
* can be a new device id or quirk for hardware, but not add major new functionality
* must already be merged into Linus’s tree

The last rule, “a change must be in Linus’s tree”, prevents the kernel community from losing fixes. The community never wants a fix to go into a stable kernel release that is not already in Linus’s tree so that anyone who upgrades should never see a regression. This prevents many problems that other projects who maintain a stable and development branch can have.

**Kernel Updates**

The Linux kernel community has promised its userbase that no upgrade will ever break anything that is currently working in a previous release. That promise was made in 2007 at the annual Kernel developer summit in Cambridge, England, and still holds true today. Regressions do happen, but those are the highest priority bugs and are either quickly fixed, or the change that caused the regression is quickly reverted from the Linux kernel tree.

This promise holds true for both the incremental stable kernel updates, as well as the larger “major” updates that happen every three months.

The kernel community can only make this promise for the code that is merged into the Linux kernel tree. Any code that is merged into a device’s kernel that is not in the kernel.org releases is unknown and interactions with it can never be planned for, or even considered. Devices based on Linux that have large patchsets can have major issues when updating to newer kernels, because of the huge number of changes between each release. SoC patchsets are especially known to have issues with updating to newer kernels due to their large size and heavy modification of architecture specific, and sometimes core, kernel code.

Most SoC vendors do want to get their code merged upstream before their chips are released, but the reality of project-planning cycles and ultimately the business priorities of these companies prevent them from dedicating sufficient resources to the task. This, combined with the historical difficulty of pushing updates to embedded devices, results in almost all of them being stuck on a specific kernel release for the entire lifespan of the device.

Because of the large out-of-tree patchsets, most SoC vendors are starting to standardize on using the LTS releases for their devices. This allows devices to receive bug and security updates directly from the Linux kernel community, without having to rely on the SoC vendor’s backporting efforts, which traditionally are very slow to respond to problems.

It is encouraging to see that the Android project has standardized on the LTS kernels as a “minimum kernel version requirement”. Hopefully that will allow the SoC vendors to continue to update their device kernels in order to provide more secure devices for their users.

**Security**

When doing kernel releases, the Linux kernel community almost never declares specific changes as “security fixes”. This is due to the basic problem of the difficulty in determining if a bugfix is a security fix or not at the time of creation. Also, many bugfixes are only determined to be security related after much time has passed, so to keep users from getting a false sense of security by not taking patches, the kernel community strongly recommends always taking all bugfixes that are released.

Linus summarized the reasoning behind this behavior in an email to the Linux Kernel mailing list in 2008:

On Wed, 16 Jul 2008, pageexec@freemail.hu wrote:

>

> you should check out the last few -stable releases then and see how

> the announcement doesn't ever mention the word 'security' while fixing

> security bugs

Umm. What part of "they are just normal bugs" did you have issues with?

I expressly told you that security bugs should not be marked as such,

because bugs are bugs.

> in other words, it's all the more reason to have the commit say it's

> fixing a security issue.

No.

> > I'm just saying that why mark things, when the marking have no meaning?

> > People who believe in them are just \_wrong\_.

>

> what is wrong in particular?

You have two cases:

- people think the marking is somehow trustworthy.

People are WRONG, and are misled by the partial markings, thinking that

unmarked bugfixes are "less important". They aren't.

- People don't think it matters

People are right, and the marking is pointless.

In either case it's just stupid to mark them. I don't want to do it,

because I don't want to perpetuate the myth of "security fixes" as a

separate thing from "plain regular bug fixes".

They're all fixes. They're all important. As are new features, for that

matter.

> when you know that you're about to commit a patch that fixes a security

> bug, why is it wrong to say so in the commit?

It's pointless and wrong because it makes people think that other bugs

aren't potential security fixes.

What was unclear about that?

Linus

This email can be found [here](http://marc.info/?l=linux-kernel&m=121616463003140), and the [whole thread](http://marc.info/?t=121507404600023) is recommended reading for anyone who is curious about this topic.

When security problems are reported to the kernel community, they are fixed as soon as possible and pushed out publicly to the development tree and the stable releases. As described above, the changes are almost never described as a “security fix”, but rather look like any other bugfix for the kernel. This is done to allow affected parties the ability to update their systems before the reporter of the problem announces it.

Linus [describes](http://marc.info/?l=linux-kernel&m=121616807207387) this method of development in the same email thread:

On Wed, 16 Jul 2008, pageexec@freemail.hu wrote:

>

> we went through this and you yourself said that security bugs are \*not\*

> treated as normal bugs because you do omit relevant information from such

> commits

Actually, we disagree on one fundamental thing. We disagree on

that single word: "relevant".

I do not think it's helpful \_or\_ relevant to explicitly point out how to

tigger a bug. It's very helpful and relevant when we're trying to chase

the bug down, but once it is fixed, it becomes irrelevant.

You think that explicitly pointing something out as a security issue is

really important, so you think it's always "relevant". And I take mostly

the opposite view. I think pointing it out is actually likely to be

counter-productive.

For example, the way I prefer to work is to have people send me and the

kernel list a patch for a fix, and then in the very next email send (in

private) an example exploit of the problem to the security mailing list

(and that one goes to the private security list just because we don't want

all the people at universities rushing in to test it). THAT is how things

should work.

Should I document the exploit in the commit message? Hell no. It's

private for a reason, even if it's real information. It was real

information for the developers to explain why a patch is needed, but once

explained, it shouldn't be spread around unnecessarily.

Linus

Full details of how security bugs can be reported to the kernel community in order to get them resolved and fixed as soon as possible can be found in the kernel file [Documentation/admin-guide/security-bugs.rst](https://www.kernel.org/doc/html/latest/admin-guide/security-bugs.html)

Because security bugs are not announced to the public by the kernel team, CVE numbers for Linux kernel-related issues are usually released weeks, months, and sometimes years after the fix was merged into the stable and development branches, if at all.

**Keeping a secure system**

When deploying a device that uses Linux, it is strongly recommended that all LTS kernel updates be taken by the manufacturer and pushed out to their users after proper testing shows the update works well. As was described above, it is not wise to try to pick and choose various patches from the LTS releases because:

* The releases have been reviewed by the kernel developers as a whole, not in individual parts
* It is hard, if not impossible, to determine which patches fix “security” issues and which do not. Almost every LTS release contains at least one known security fix, and many yet “unknown”.
* If testing shows a problem, the kernel developer community will react quickly to resolve the issue. If you wait months or years to do an update, the kernel developer community will not be able to even remember what the updates were given the long delay.
* Changes to parts of the kernel that you do not build/run are fine and can cause no problems to your system. To try to filter out only the changes you run will cause a kernel tree that will be impossible to merge correctly with future upstream releases.

Note, this author has audited many SoC kernel trees that attempt to cherry-pick random patches from the upstream LTS releases. In every case, severe security fixes have been ignored and not applied.

As proof of this, I demoed at the Kernel Recipes talk referenced above how trivial it was to crash all of the latest flagship Android phones on the market with a tiny userspace program. The fix for this issue was released 6 months prior in the LTS kernel that the devices were based on, however none of the devices had upgraded or fixed their kernels for this problem. As of this writing (5 months later) only two devices have fixed their kernel and are now not vulnerable to that specific bug.

Posted by Greg Kroah-Hartman Feb 5th, 2018  [kernel](http://kroah.com/log/blog/categories/kernel/), [linux](http://kroah.com/log/blog/categories/linux/), [stable](http://kroah.com/log/blog/categories/stable/)