An engineer’s guide to Linux Kernel upgrades

Ignat Korchagin
@ignatkn
$ whoami

- Linux team at Cloudflare
- Systems security and performance
- Low-level programming
What do you do in this case?
Updates available!

Updated software is available for this computer. Do you want to install it now?

Details of updates

177.7 MB will be downloaded.

Settings... Remind Me Later Install Now
Updates available for production systems!

remind me later

disable updates

@ignatkn
How do we perceive software updates?
Software updates perception

Regular software upgrades
Software updates perception

Regular software upgrades

Linux Kernel upgrades

@ignatkn
Regular software updates

Segmentation fault
Regular software updates

Segmentation fault

systemd service unit file

... [Service]
Restart=always
...

...
Regular software updates

Segmentation fault

systemd service unit file

...  
[Service]  
Restart=always  
...
Linux Kernel updates
Common risks of not applying software updates

And Linux Kernel in particular
Bugs are not getting fixed
Bugs are not getting fixed

Commits per release for 6.1.x branch
Bugs are not getting fixed

Commits per release for 6.1.x branch

Out of 76 releases:
- 50 with $\geq 100$ commits
- 20 with $\geq 200$ commits
- 5 with $\geq 500$ commits
Missing out on performance improvements
Missing out on performance improvements
Linux 5.4 to 5.10 migration
Missing out on performance improvements
Linux 5.4 to 5.10 migration
Missing out on performance improvements

Linux 5.4 to 5.10 migration: saved ~4.5 GiB of RAM per server
Missing out on performance improvements

Linux 5.4 to 5.10 migration: saved ~4.5 GiB of RAM per server

https://patchwork.kernel.org/project/linux-mm/cover/20191018002820.307763-1-guro@fb.com/
Accumulating change delta

Total commits per release for 6.1.x branch
Accumulating change delta

Change delta (risk):
- 6.1.10 vs 6.1.20: 1762

Total commits per release for 6.1.x branch
Accumulating change delta

Total commits per release for 6.1.x branch

Change delta (risk):
- 6.1.10 vs 6.1.20: 1762
- 6.1.10 vs 6.1.30: 3910
Accumulating change delta

Change delta (risk):
- 6.1.10 vs 6.1.20: 1762
- 6.1.10 vs 6.1.30: 3910
- \( \frac{3910}{1762} = \sim 2.21 \)
Accumulating change delta

Change delta (risk):
- 6.1.10 vs 6.1.20: 1762
- 6.1.10 vs 6.1.30: 3910
- $\frac{3910}{1762} \approx 2.21$
- for 2x delay we get $\approx 2.21$ more risk!
Security vulnerabilities are not getting fixed
Security vulnerabilities are not getting fixed

CVEs fixed per release for 6.1.x branch

source: https://www.linuxkernelcves.com
Security vulnerabilities are not getting fixed

Out of 71 releases:
- 56 with ≥ 1 CVE patched
- 18 with ≥ 5 CVEs patched

source: https://www.linuxkernelcves.com
Compliance risks
Compliance risks

6.3.3 All system components are protected from known vulnerabilities by installing applicable security patches/updates as follows:

- Critical or high-security patches/updates (identified according to the risk ranking process at Requirement 6.3.1) are installed within **one month of release**.

- All other applicable security patches/updates are installed within an appropriate time frame as determined by the entity (for example, within three months of release).
Compliance risks

Remember?
(Not so)fun fact: if your uptime >= 30 days, you’re system is likely vulnerable!
Common anti patterns for Linux Kernel releases
Which things from the changelog are applicable to us?
Let’s justify the upgrade

Out of 76 releases:

- 50 with $\geq 100$ commits
- 20 with $\geq 200$ commits
- 5 with $\geq 500$ commits
Let’s justify the upgrade
Is this security vulnerability actually exploitable on our systems?
Is this vulnerability applicable to us?

The attacker

- Highly motivated to break into the system
- Spends exclusively almost 24/7 to design and implement a successful exploit
Is this vulnerability applicable to us?

The attacker

- Highly motivated to break into the system
- Spends exclusively almost 24/7 to design and implement a successful exploit
Is this vulnerability applicable to us?

**The attacker**
- Highly motivated to break into the system
- Spends exclusively almost 24/7 to design and implement a successful exploit

**Security patch reviewer**
- Highly motivated to go home on time
- Needs to review several patches a day
- Has other competing priorities
Is this vulnerability applicable to us?

The attacker
- Highly motivated to break into the system
- Spends exclusively almost 24/7 to design and implement a successful exploit

Security patch reviewer
- Highly motivated to go home on time
- Needs to review several patches a day
- Has other competing priorities
Let it soak

Let’s soak it for 1 month in canary to ensure it is stable
Let it soak

Change delta (risk):
- 6.1.10 vs 6.1.20: 1762
- 6.1.10 vs 6.1.30: 3910
- 3910/1762 = ~2.21
- for 2x delay we get ~2.21 more risk!
Let it soak

Total commits per release for 6.1.x branch

CVEs fixed per release for 6.1.x branch

Change delta (risk):
- 6.1.10 vs 6.1.20: 1762
- 6.1.10 vs 6.1.30: 3910
- $\frac{3910}{1762} = \sim 2.21$
- for 2x delay we get $\sim 2.21$ more risk!

Out of 71 releases:
- 56 with $\geq$ 1 CVE patched
- 18 with $\geq$ 5 CVEs patched
Let it soak

Total commits per release for 6.1.x branch

CVEs fixed per release for 6.1.x branch

Change delta (risk):
- 6.1.10 vs 6.1.20: 1762
- 6.1.10 vs 6.1.30: 3910
- $\frac{3910}{1762} = \sim 2.21$
- for 2x delay we get $\sim 2.21$ more risk!

Out of 71 releases:
- 56 with $\geq 1$ CVE patched
- 18 with $\geq 5$ CVEs patched
Let it soak

High “soak” times probably means

- We don’t know what we are looking for
  - Lack of metrics/observability
Let it soak

High “soak” times probably means

- We don’t know what we are looking for
  - Lack of metrics/observability
- We don’t know our workload
  - What kernel features/subsystems are important to us
Let it soak

High “soak” times probably means

- We don’t know what we are looking for
  - Lack of metrics/observability
- We don’t know our workload
  - What kernel features/subsystems are important to us
- Lack of sufficient pre-production kernel testing
  - Unit tests
  - Integration tests
  - Performance tests
The Kernel is too critical! Let’s have more approvals before the deploy!
The Kernel is too critical! Let’s have more approvals before the deploy!
Too risky!

WHAT IF I TOLD YOU

KERNEL DEPLOYS ARE INHERENTLY SAFER THAN OTHER SOFTWARE
Automated software deploys
Automated software deploys

Regular software

- Upgrade software package
Automated software deploys

Regular software

- Upgrade software package
- Service restart
  - graceful/non-graceful
Automated software deploys

Regular software

- Upgrade software package
- Service restart
  - graceful/non-graceful
- New (bad or good) code can propagate to production in minutes without appropriate safeguards
Automated software deploys

Regular software
- Upgrade software package
- Service restart
  - graceful/non-graceful
- New (bad or good) code can propagate to production in minutes without appropriate safeguards

Linux Kernel
- Requires a reboot
  - Drain traffic from the server
  - Put it out of production
  - Reboot
  - Wait for it to be re-configured
  - Run acceptance tests
  - Put back in production
Automated software deploys

Regular software

- Upgrade software package
- Service restart
  - graceful/non-graceful
- **New (bad or good) code can propagate to production in minutes without appropriate safeguards**

Linux Kernel

- Requires a reboot
  - Drain traffic from the server
  - Put it out of production
  - Reboot
  - Wait for it to be re-configured
  - Run acceptance tests
  - Put back in production
  - We don’t reboot all servers at once
Automated software deploys

Regular software
- Upgrade software package
- Service restart
  - graceful/non-graceful
- New (bad or good) code can propagate to production in minutes without appropriate safeguards

Linux Kernel
- Requires a reboot
  - Drain traffic from the server
  - Put it out of production
  - Reboot
  - Wait for it to be re-configured
  - Run acceptance tests
  - Put back in production
  - We don’t reboot all servers at once
  - Inherently slow-paced gradual rollout with minimal impact, if things go wrong
Linux Kernel releases explained

Not every kernel release is created equal
Kernel release numbers

X.XX.XX
Kernel release numbers

X.XX.XX
(ex 6.1.32)
Kernel release numbers

X.XX.XX
(ex 6.1.32)

https://semver.org/
Kernel release numbers

X.XX.XX
(ex 6.1.32)

But it is NOT a semver!

https://semver.org/
Kernel release numbers

X.XX.XX
Kernel release numbers

\[ X.xxx.xx \]

Major version
Kernel release numbers

X.XX.XX

Major version

(NOT major/minor)
Kernel release numbers

X.XXX.XXX

- Major version
- Bugs and security fixes

(NOT major/minor)
Kernel release numbers

X.XX.XX

- Major version
  - (NOT major/minor)
- Bugs and security fixes
- Never new features or major subsystem rewrites
Kernel release flow
Kernel release flow

@ignatkn

torvalds/linux.git
Kernel release flow

drivers

merge

mm

merge

net

torvalds/linux.git
Kernel release flow

- drivers
- mm
- net

merge

torvalds/linux.git

branch out

6.1
6.2
6.3
Kernel release flow

- drivers
- mm
- net

merge

branch out

torvalds/linux.git

6.1
6.2
6.3

9-10 weeks
Kernel release flow

- **drivers**
- **mm**
- **net**

Merge flow:
- drivers to mm
- mm to net

Branch flow:
- 6.1 branches out to 6.2 and 6.3
- 6.2 branches out to 6.2.1

Release flow:
- 6.1 is tagged after 9-10 weeks.
Kernel release flow

- drivers
- mm
- net

- merge
- merge
- branch out
- branch out

- bugfix

- 6.1
- 6.2
- 6.3

- 9-10 weeks

- tagged

- 6.2.1
Kernel release flow

- drivers
- mm
- net
- bugfix
- torvalds/linux.git
- 6.1
- 6.2
- 6.3
- 6.2.1

Branch out 9-10 weeks
Merge
Merge
Tagged
Kernel release flow
Kernel release flow

- **drivers**
- **mm**
- **net**

**bugfix** 

- **torvalds/linux.git**

**merge** 

- **branch out**
- **cherry pick**

**6.1** 
**6.2** 
**6.3**

- **9-10 weeks**
- **every week**

**tagged** 

- **6.2.1** 
- **6.2.2**
Linux Kernel releases

- A new major (stable) kernel version is released every 9-10 weeks
  - 2 weeks for development/7 weeks for bugfixing
Linux Kernel releases

- A new major (stable) kernel version is released every 9-10 weeks
  - 2 weeks for development/7 weeks for bugfixing
- Leftmost version number **means nothing**
  - 4.19.x → 4.20.x upgrade can contain more features/breaking changes than 4.20.x → 5.0.x
Linux Kernel releases

- A new major (stable) kernel version is released every 9-10 weeks
  - 2 weeks for development/7 weeks for bugfixing
- Leftmost version number **means nothing**
  - $4.19.x \rightarrow 4.20.x$ upgrade can contain more features/breaking changes than $4.20.x \rightarrow 5.0.x$
- Bugfix/patch releases are released around once a week
  - Denoted by rightmost version number
  - Usually cherry-picked from the main Linux branch
  - No new features, therefore regressions are quite rare
  - May contain critical security patches
  - You **almost always** want to apply them
Longterm releases

- Usually a stable release branch is active around 2-3 months
  - After that it is EOL and no bugfixes are backported (including critical security vulnerabilities)
  - A new major stable version should be available at this point
Longterm releases

- Usually a stable release branch is active around 2-3 months
  - After that it is EOL and no bugfixes are backported (including critical security vulnerabilities)
  - A new major stable version should be available at this point
- But there are “longterm” stable releases
  - Bug and security fixes are backported for at least 2 years
  - Usually the last stable release of the year
    - Therefore, released once a year
  - Provides enough time for more rigid evaluation of the next “longterm” release
Longterm releases

- Usually a stable release branch is active around 2-3 months
  - After that it is EOL and no bugfixes are backported (including critical security vulnerabilities)
  - A new major stable version should be available at this point
- But there are “longterm” stable releases
  - Bug and security fixes are backported for at least 2 years
  - Usually the last stable release of the year
    - Therefore, released once a year
  - Provides enough time for more rigid evaluation of the next “longterm” release

https://www.kernel.org/category/releases.html
Safe and easy production kernel upgrades
Safe and easy production kernel upgrades

Don’t create a dedicated deploy process for the Linux Kernel
Safe and easy production kernel upgrades

Don’t create a dedicated deploy process for the Linux Kernel

- Kernel upgrades are usually less risky than other software
- A simple staged rollout is usually enough
- Kernel upgrades are naturally slow paced, because they require a reboot
  - A lot of headroom to abort the deploy if things look wrong
Safe and easy production kernel upgrades

Avoid justifying a bugfix kernel upgrades
Safe and easy production kernel upgrades

Avoid justifying a bugfix kernel upgrades

- Should be released with “no questions asked”
- Contain only bug fixes and security patches
  - And most likely some are always applicable
- Regressions are quite uncommon
- Minimise canary “soak” times
  - Use metrics-driven approach instead
Safe and easy production kernel upgrades

Stay on the “longterm” branch, if validating a major version is costly
Safe and easy production kernel upgrades

Stay on the “longterm” branch, if validating a major version is costly

- At least two years of bugfixes and security patches
- But start evaluating the next “longterm” release early in ~1 year
  - More features
  - Better performance and resource utilisation
- Accumulating less change delta
Safe and easy production kernel upgrades

Implement/improve pre-production testing for major version validation
Safe and easy production kernel upgrades

Implement/improve pre-production testing for major version validation

- Understand your workload
- Write tests, which exercise various kernel subsystems required by your workload
  - Can help when communicating issues to the kernel community
- Make metrics-driven decisions
  - Not time-based decisions (minise “soak” times)
Safe and easy production kernel upgrades

Metrics, monitoring and deploy automation can help with human risk perception
Safe and easy production kernel upgrades

Metrics, monitoring and deploy automation can help with human risk perception

- Data-driven decision if the deploy looks good
- Provides quick early signals about regressions
- Can save the engineering team a debugging cycle
- Automation encourages regular upgrades
  - Removes the need for an operator to perform a “potentially risky” release
Conclusions

- Linux Kernel upgrades are not more risky than any other software
- You need to patch early and patch often
- Bugfix kernel releases should be applied with “no questions asked”
- Understanding your workload, metrics, monitoring and automation allow your systems to stay patched and secure
Thank you!

Questions?