What does log4j teach us about the software supply chain?

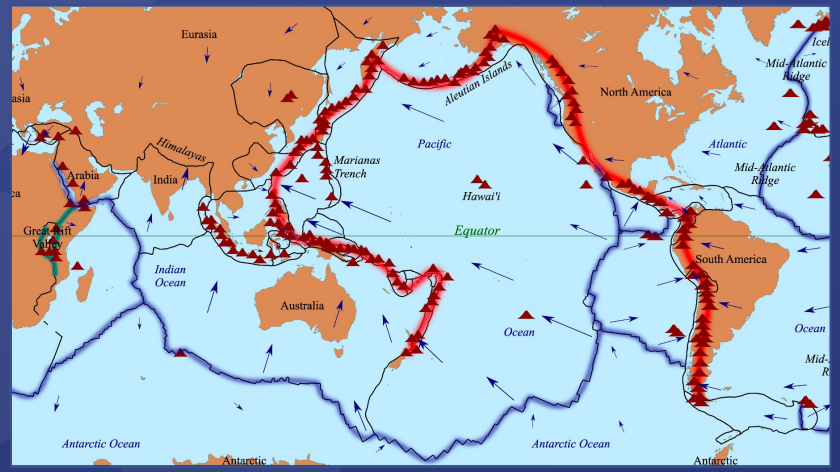
Dr. Stephen Magill

VP of Product Innovation

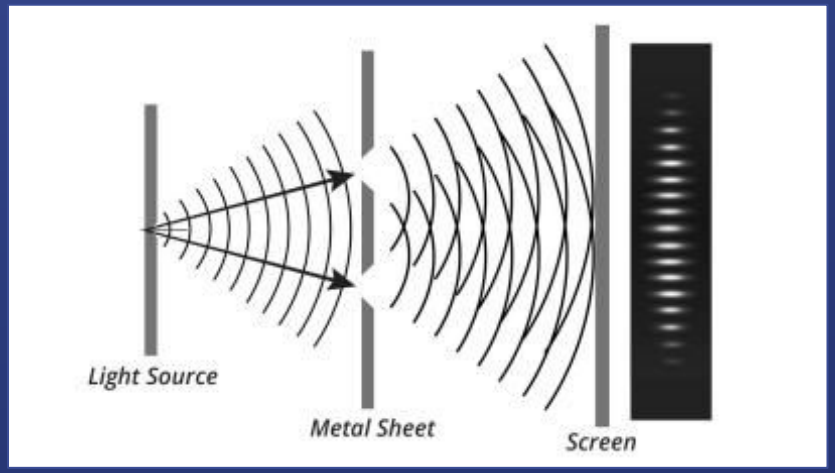

Sonatype
2019
State
Softv
Sup
Chain
The 5th annual report o
open source software d
presented by
sonatype
2020
State of the
Software
Supply Chain
The 6th Annual Report on Global Open Source Software Development
PRESENTED BY
sonatype
IN PARTNERSHIP WITH
IT REVOLUTION
muse dev


Sonatype
2020
State of the Software Supply Chain
The 6th Annual Report on Global Open Source Software Development
PRESENTED BY
sonatype
IN PARTNERSHIP WITH
IT REVOLUTION
muse dev
log4shell

Plate Tectonics



Quantum Mechanics

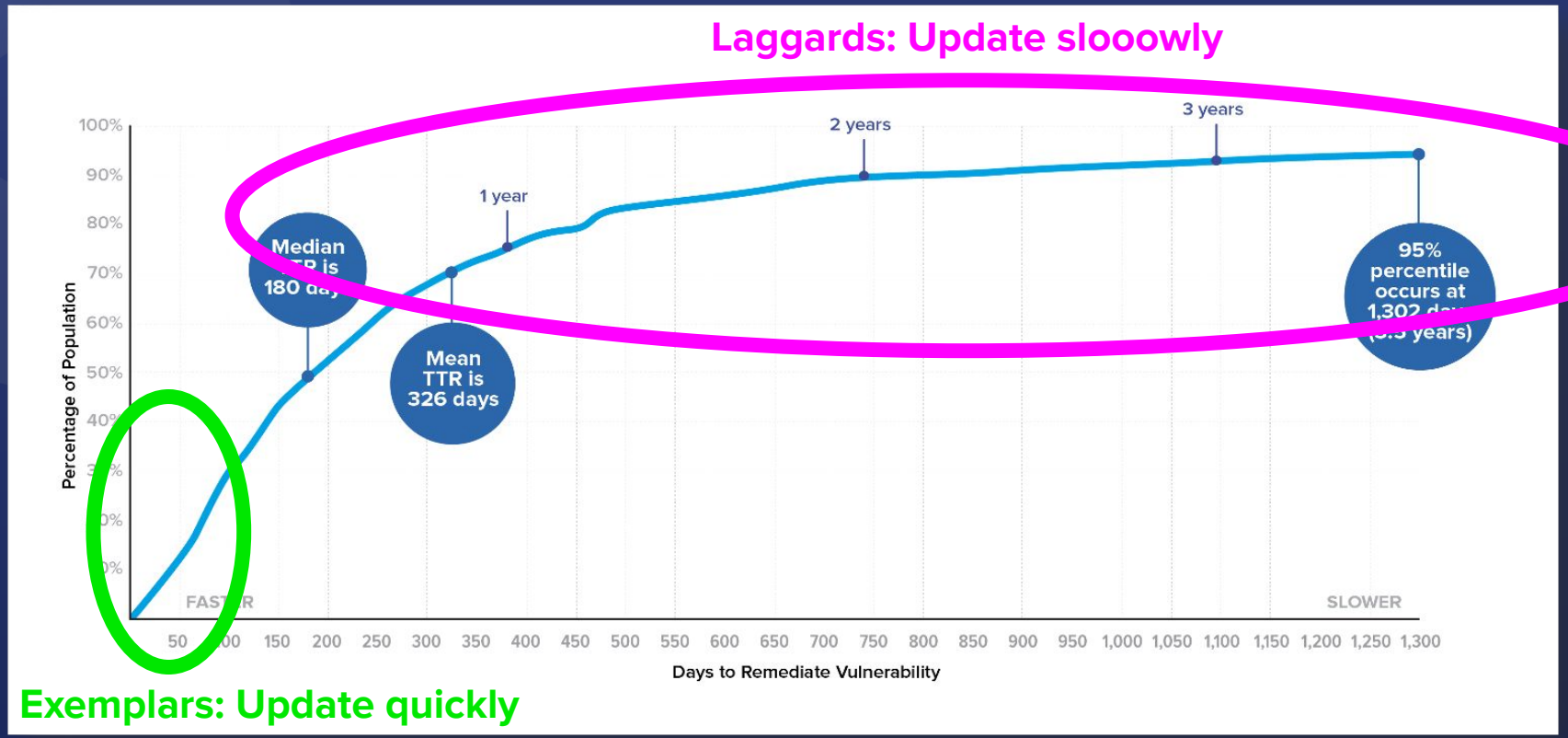


OR

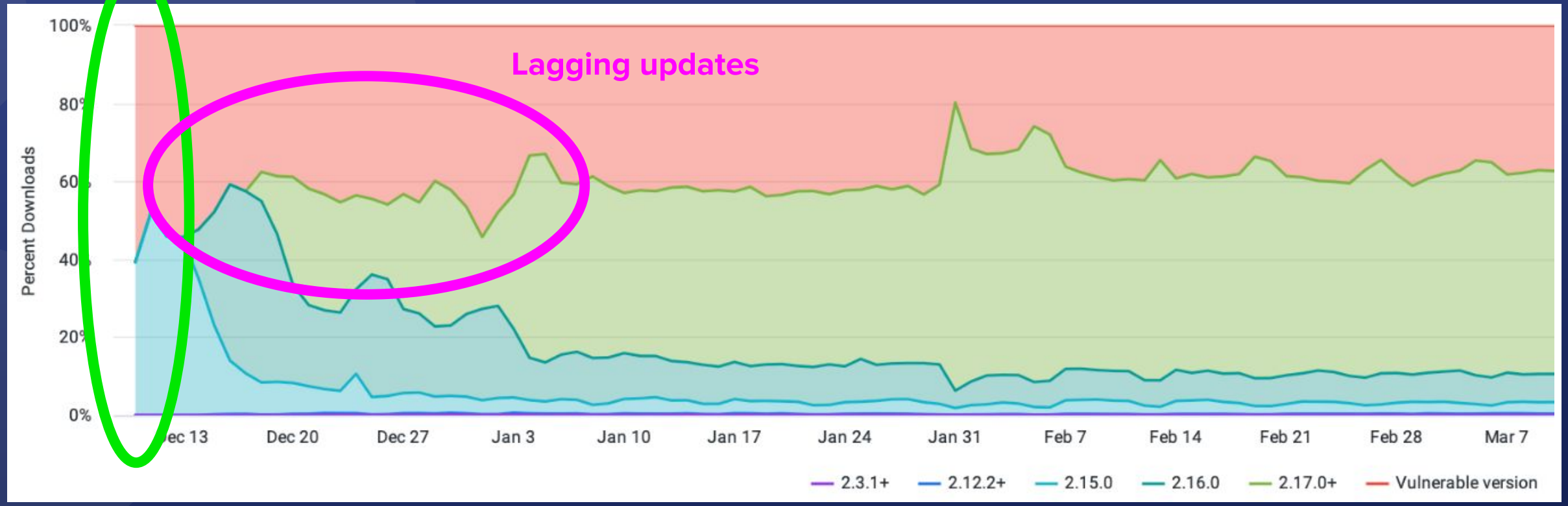
CERN LHC

Confirms much of what we already knew

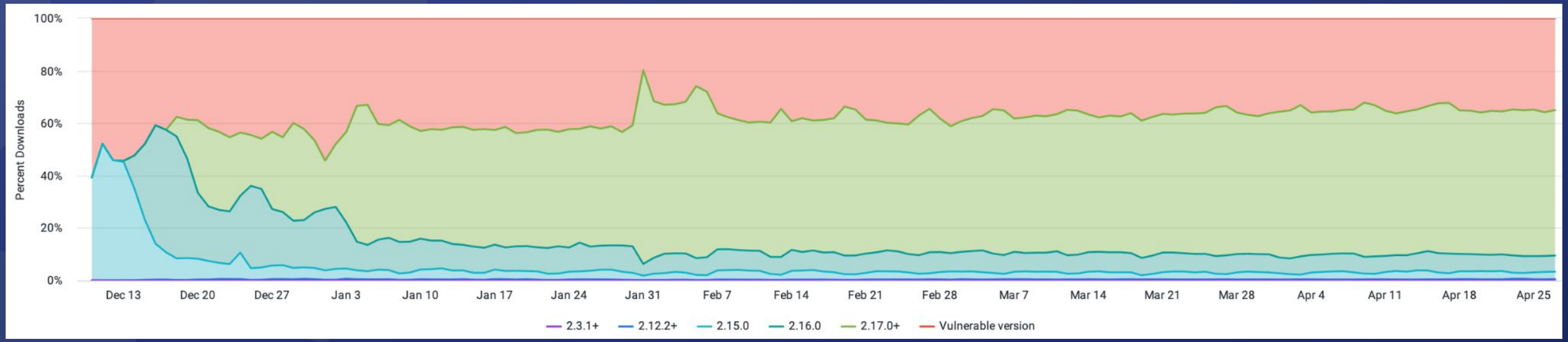
Concept 1:
Exemplars and Laggards



log4j Download Percentages by Version

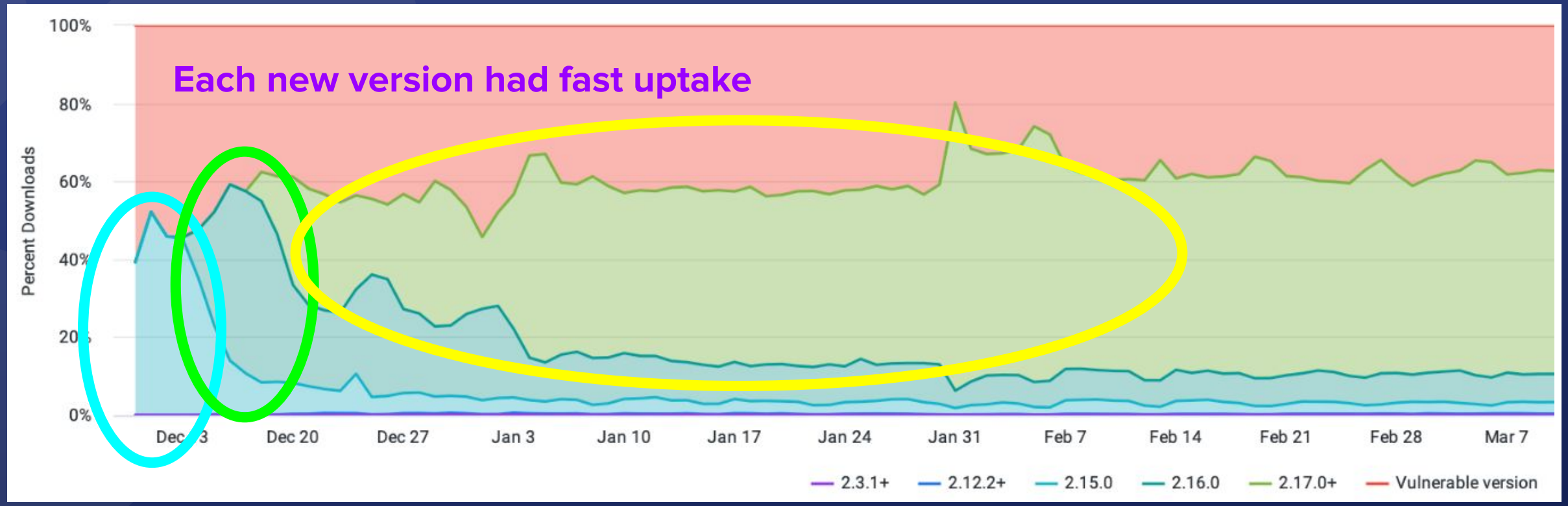


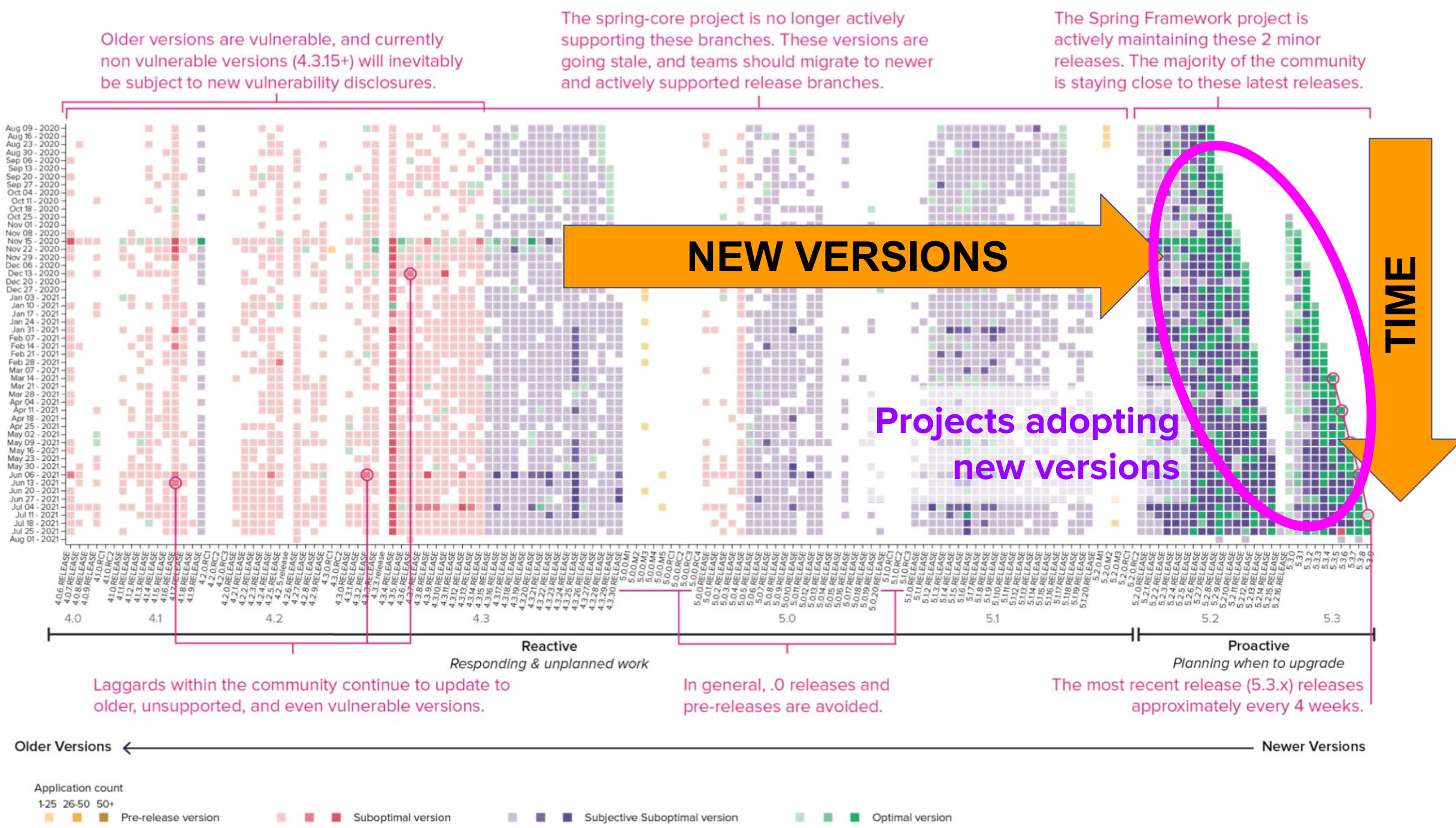
Exemplars: 1-2 day update window

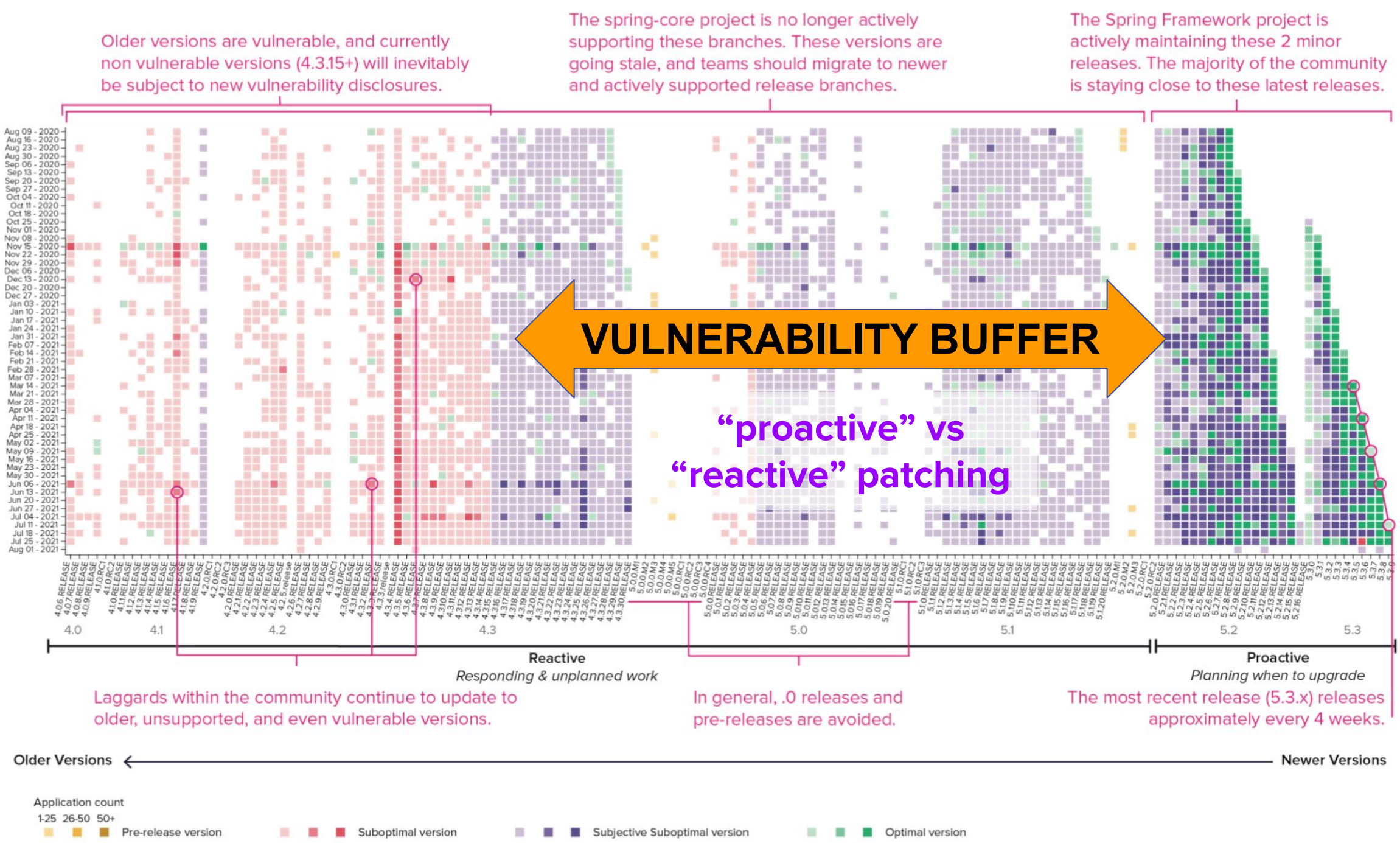

100%
80%
60%
40%
20%
0%
Percent Downloads
Dec 13
Dec 20
Dec 27
Jan 3
Jan 10
Jan 17
Jan 24
Jan 31
Feb 7
Feb 14
Feb 21
Feb 28
Mar 7
Mar 14
Mar 21
Mar 28
Apr 4
Apr 11
Apr 18
Apr 25
2.3.1+
2.12.2+
2.15.0
2.16.0
2.17.0+
Vulnerable version

Concept 2:
Staying Secure by Staying Up-to-date

log4j Download Percentages by Version




Older versions are vulnerable, and currently non vulnerable versions (4.3.15+) will inevitably be subject to new vulnerability disclosures.
The spring-core project is no longer actively supporting these branches. These versions are going stale, and teams should migrate to newer and actively supported release branches.
The Spring Framework project is actively maintaining these 2 minor releases. The majority of the community is staying close to these latest releases.
NEW VERSIONS
TIME
Projects adopting new versions
4.0
4.1
4.2
4.3
5.0
5.1
5.2
5.3
Reactive
Responding & unplanned work
Proactive
Planning when to upgrade
Laggards within the community continue to update to older, unsupported, and even vulnerable versions.
In general, .0 releases and pre-releases are avoided.
The most recent release (5.3.x) releases approximately every 4 weeks.
Older Versions
Newer Versions
Application count
1-25 26-50 50+
Pre-release version
Suboptimal version
Subjective Suboptimal version
Optimal version


Older versions are vulnerable, and currently non vulnerable versions (4.3.15+) will inevitably be subject to new vulnerability disclosures.
The spring-core project is no longer actively supporting these branches. These versions are going stale, and teams should migrate to newer and actively supported release branches.
The Spring Framework project is actively maintaining these 2 minor releases. The majority of the community is staying close to these latest releases.
VULNERABILITY BUFFER
"proactive" vs
"reactive" patching
4.0
4.1
4.2
4.3
5.0
5.1
5.2
5.3
Reactive
Responding & unplanned work
Proactive
Planning when to upgrade
Laggards within the community continue to update to older, unsupported, and even vulnerable versions.
In general, .0 releases and pre-releases are avoided.
The most recent release (5.3.x) releases approximately every 4 weeks.
Older Versions
Newer Versions
Application count
1-25 26-50 50+
Pre-release version
Suboptimal version
Subjective Suboptimal version
Optimal version

Concept 3:
Transitive Dependencies Matter

- Repositories 146
- Code 0
- Commits 2K+
- Issues 4K
- Discussions 87
- Packages 4
- Marketplace 0
- Topics 1
- Wikis 42
- Users 1

Advanced search Cheat sheet

Showing 2,186 available commit results

Sort: Least recently committed

genny-project/qwanda-services
bump log4j to 2.15.0 to fix CVE-2021-44228
hbjoylee committed on Dec 9, 2021
Verified 6ac6bcb

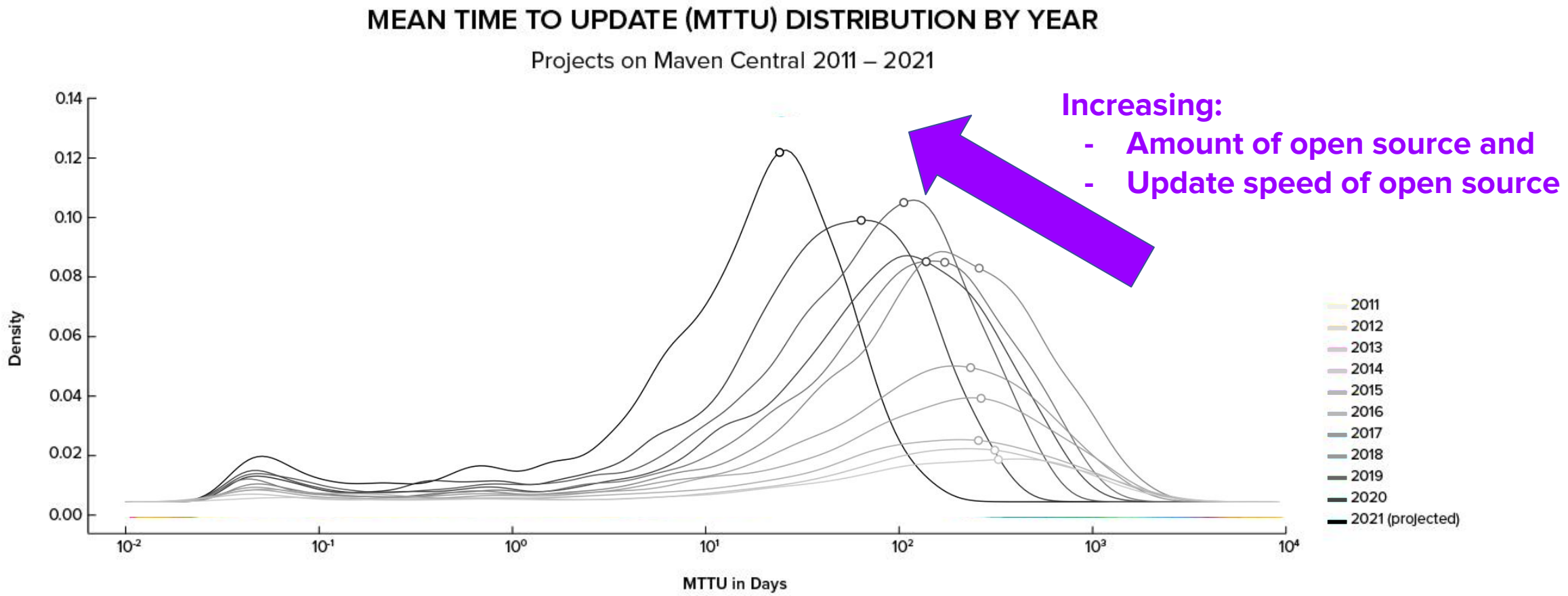
genny-project/messages
bump log4j to 2.15.0 to fix CVE-2021-44228
hbjoylee committed on Dec 9, 2021
Verified 8eeb82c

MayBlock/Terminal
修复Log4j2 漏洞及其他漏洞 ...
MayBlock committed on Dec 9, 2021
Verified 2097172

MayBlock/Terminal
修复Log4j2 漏洞及其他漏洞（重要） ...
MayBlock committed on Dec 9, 2021
Verified 8404549

MayBlock/Terminal
修复Log4j2 漏洞及其他漏洞 ...
MayBlock committed on Dec 9, 2021
Verified 4fa5b5e

Update Speed Consistently Improving Year-Over-Year



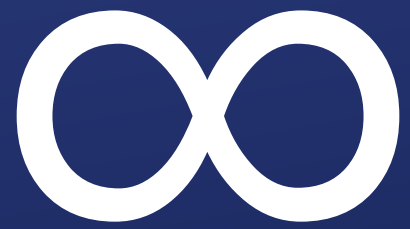
Increasing:

- **Amount of open source and**
- **Update speed of open source**

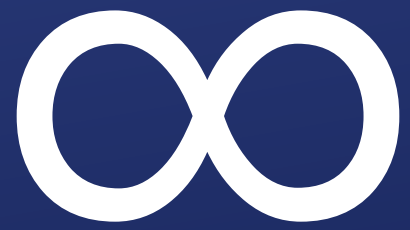
SOURCE: 2021 STATE OF THE SOFTWARE SUPPLY CHAIN REPORT BY SONATYPE

Concept 4:
Some dependencies just never get upgraded

How long did it take to get to 90% remediation?



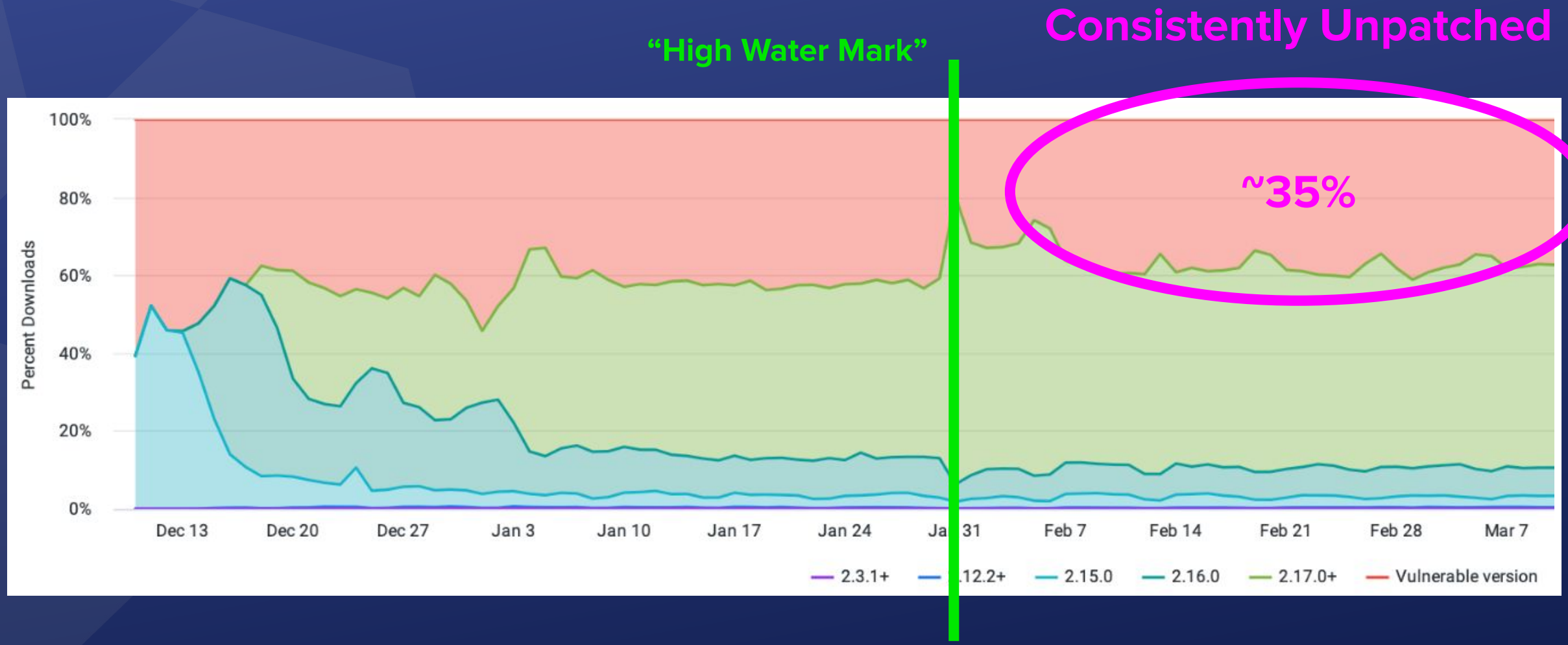
How long did it take to get to 80% remediation?



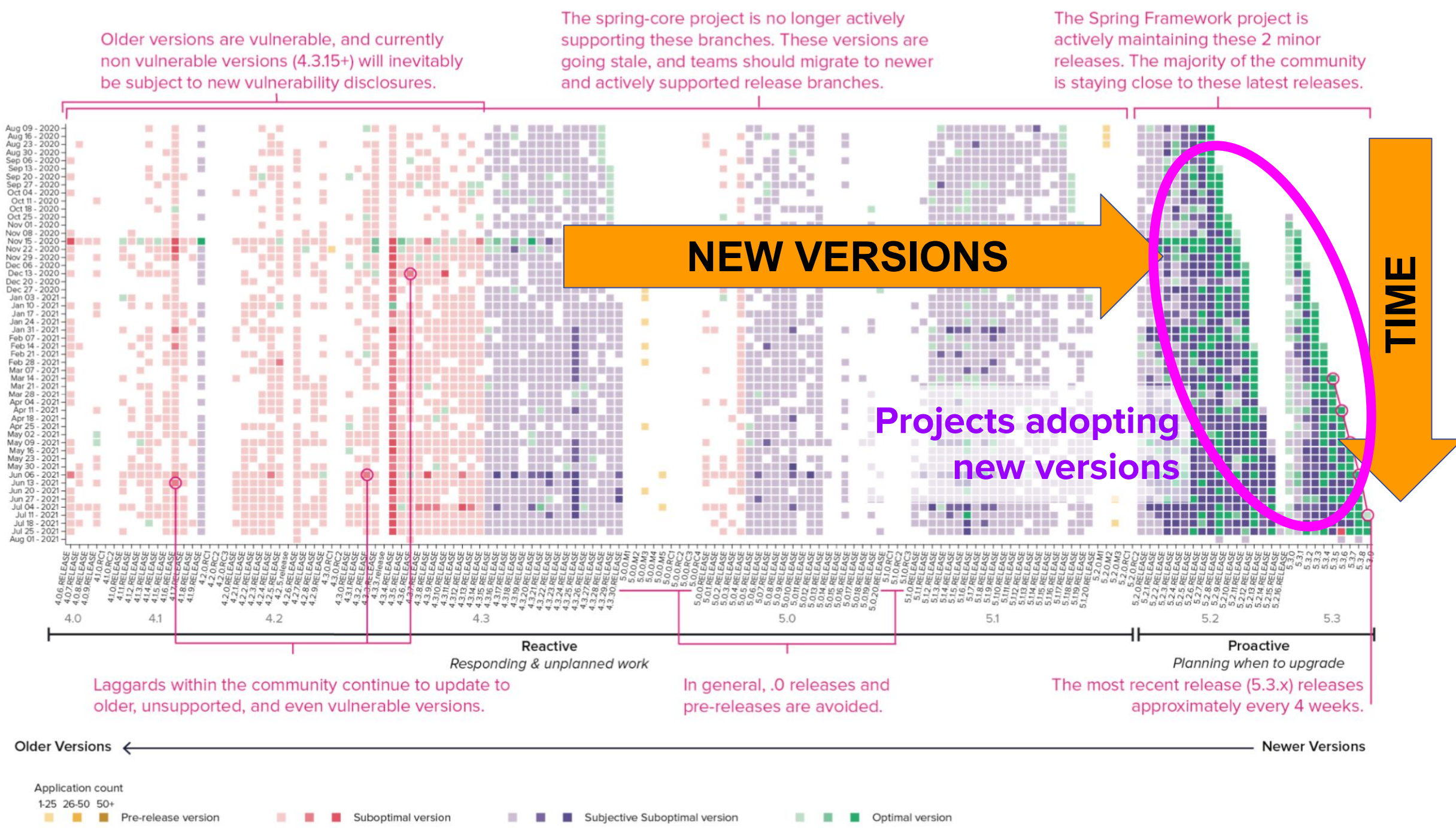
How long did it take to get to 70% remediation?

52 days

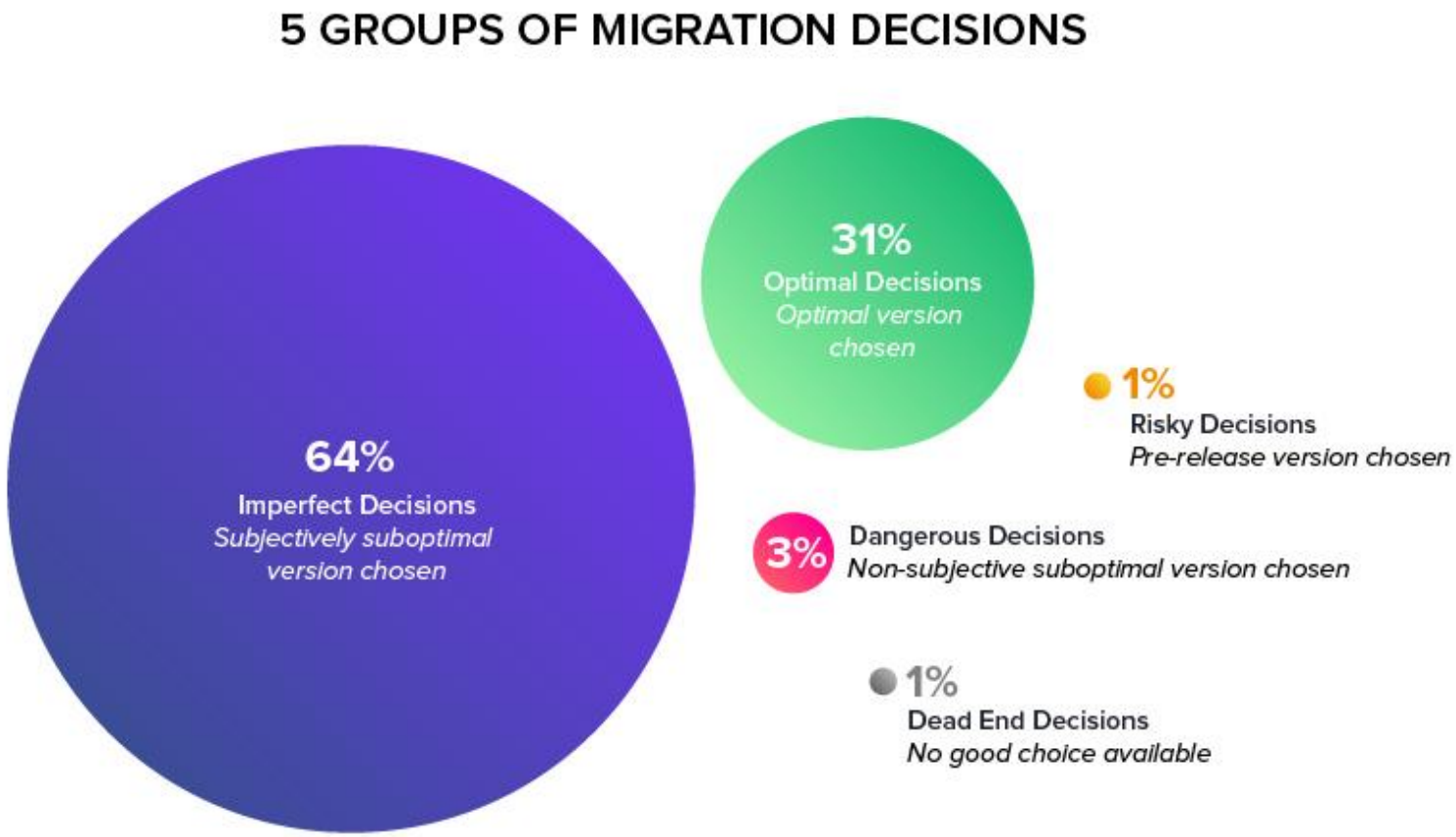
log4j Download Percentages by Version



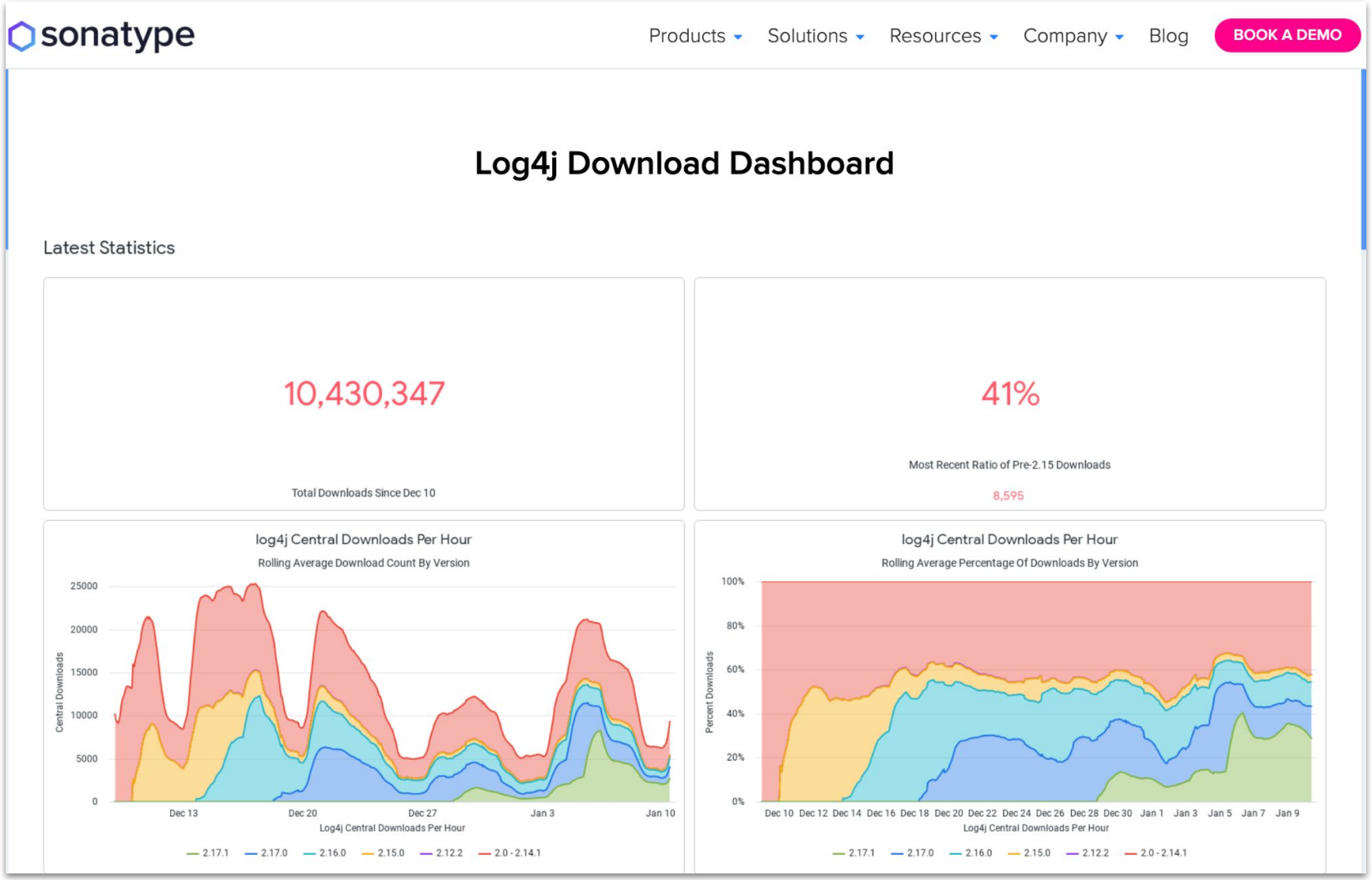
2021 Supply Chain Report Found: 75% of Dependencies Were Never Upgraded


Older versions are vulnerable, and currently non vulnerable versions (4.3.15+) will inevitably be subject to new vulnerability disclosures.
The spring-core project is no longer actively supporting these branches. These versions are going stale, and teams should migrate to newer and actively supported release branches.
The Spring Framework project is actively maintaining these 2 minor releases. The majority of the community is staying close to these latest releases.
NEW VERSIONS
TIME
Projects adopting new versions
4.0
4.1
4.2
4.3
5.0
5.1
5.2
5.3
Reactive
Responding & unplanned work
Proactive
Planning when to upgrade
Laggards within the community continue to update to older, unsupported, and even vulnerable versions.
In general, .0 releases and pre-releases are avoided.
The most recent release (5.3.x) releases approximately every 4 weeks.
Older Versions
Newer Versions
Application count
1-25 26-50 50+
Pre-release version
Suboptimal version
Subjective Suboptimal version
Optimal version

69% of dependency management decisions are suboptimal.



Despite disclosure, almost half of downloads were vulnerable 1 Month Later



https://www.sonatype.com/resources/log4j-vulnerability-resource-center

Latest Statistics

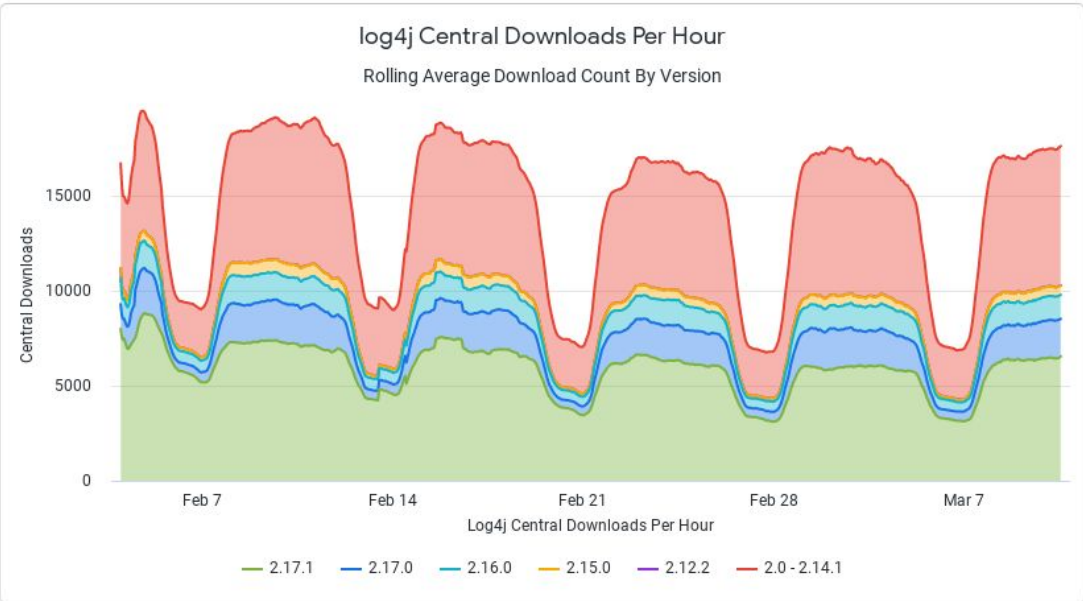
31,403,835

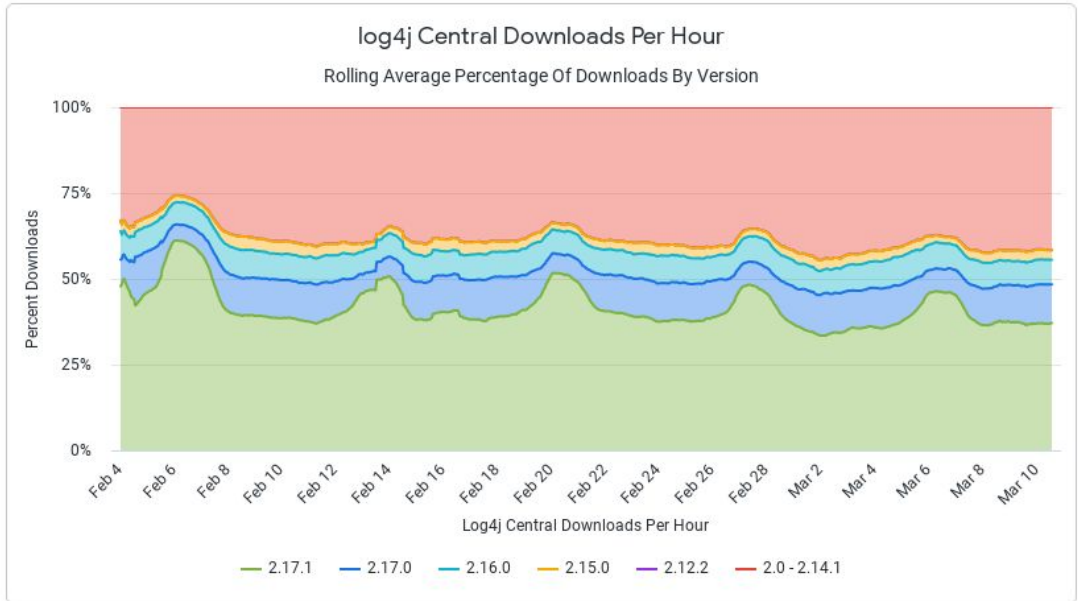
Total Downloads Since Dec 10

38%

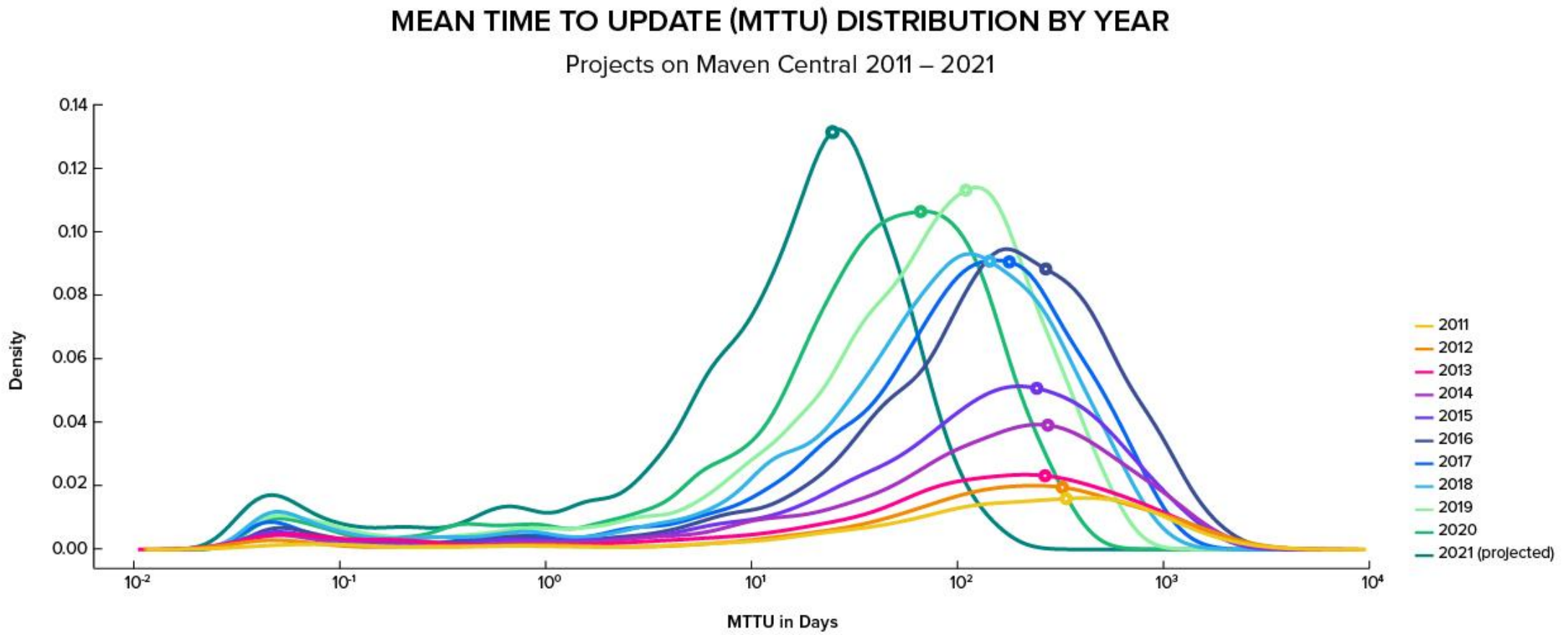
Most Recent Ratio of Pre-2.15 Downloads

8,387





Aggregate MTTUs are improving over time.



SOURCE: 2021 STATE OF THE SOFTWARE SUPPLY CHAIN REPORT BY SONATYPE

Takeaways

- Stay secure by staying **up to date.**
- As open source update performance improves this becomes **more and more** effective (transitive dependencies).
- Make sure you're updating **all** your dependencies.
- Be an **exemplar!**

Additional Guidance for Zero-days

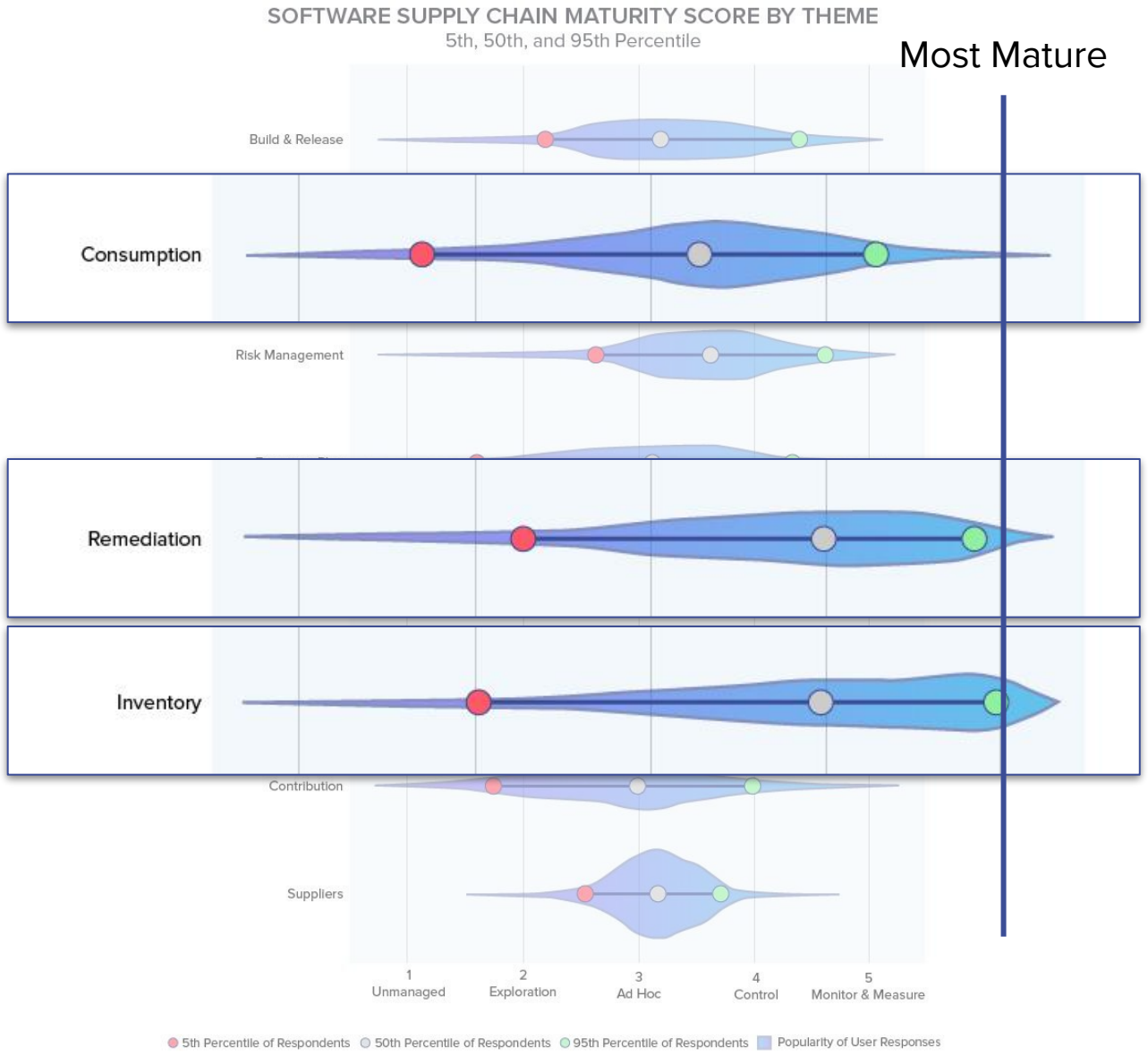
(like log4j)

What To Do? A remediation step-by-step.

Inventory: Full Software Bill Of Materials for all applications

Consumption: Ability to monitor and approve / block incoming components; Process for approving components

Remediation: Continuous monitoring and remediation advice integrated with development workflows; DevOps and CI/CD



Thank You!

Log4j

With automation:

- 6 days to 50% remediated
- 1/4 65% remediated

Without automation:

- 11 days to 50%

We did these SSCR reports and then log4j happened

Did it confirm what we had seen or obliterate our carefully-constructed hypotheses?

It confirmed what we had seen

Let me first tell you what we saw in log4j, then how it's consistent with what we've seen in the report, then close with some advice from that research on what can help with the next log4j.

log4j was a stress test of our ability to secure the software supply chain. and I mean stress test in a very literal sense – it was stressful for individuals all over the world and we'll hear about a particularly compelling on-the-ground story next from ____ from Morgan Stanley

Does log4j prove that the community is great about security?

Does it prove we're terrible at security?

Both are true

Time to 90% remediation? infinity

80%? infinity

etc.

that shows bad

but security community was great about quickly and transparently performing research to find vulnerabilities and patch them

series of patches

uptake of the series was good!

but uptake overall was bad

what this shows is that when we upgrade, we're pretty good at it. not perfect though (show old version uptake increasing). so how "not perfect" and how "good"? this is research so we need to quantify those.

show bubbles

but we only do the 25% of the time. and this is where "less than perfect" matters. if we could be more efficient we could take the same amount of update time and stay on top of 50% or 75% of our dependencies.

And that helps us stay ahead of vulnerabilities. Because log4j was an anomaly. a 0-day. usually patches are out well ahead of disclosure. this means if you just stay up to date, you stay secure.

show migration chart and the creeping wall of red. talk about proactive vs reactive.

Another thing to note is that even if you are great about being at the right, that's not the whole story. because your dependencies have dependencies, you're reliant on them to update and fix issues in transitive dependencies. but there's good news on this front. the community has been getting better at this. show decreasing MTTU chart.

So the takeaways:

- community is there
- you just have to do it
- do it efficiently and you can cover more of your risk surface

Last 5 minutes

And what do you do if there's another 0-day? staying up-to-date doesn't help there. you can't be proactive, you have to be reactive.

so what helps you react faster? talk about remediation steps.

central inventory

automation that is "ready to go" – push-button ability to scan apps and block releases containing vulnerable versions

curation process for dependencies. some criteria to ensure they will be responsive like the log4j authors. also some process to ensure you don't end up with some of EVERY logging library. because then you increase your work a bunch.

keep it constrained, keep it catalogued, keep it uniform and above all, keep it automated