Sourcegraph

How *Google*, *Microsoft*, *Lyft*, *GitLab*, and *Atlassian* find and fix vulnerabilities

If you have a codebase, then you have to find and fix vulnerabilities. A codebase is only as robust as its code security program. This fact unites tech companies large and small, from the biggest enterprises to the smallest startups.

When, for example, a vulnerability in Log4j emerged in December 2021, everyone from tech enterprises like Google and Microsoft to commercial companies and startups had to find and fix every impacted version of Lo4j in their codebases. Some companies deployed detailed vulnerability management and incident response plans; some scrambled.

Not every vulnerability will be quite so urgent, but Log4j did show many companies that their vulnerability management and code security processes needed improvement. In this article, we're exploring ways Google, Microsoft, Lyft, GitLab, and Atlassian find and fix vulnerabilities. With their lessons learned, you can hone your vulnerability management process and improve your code security posture.

Google uses static analysis and large-scale changes to find and fix vulnerabilities

Google has a multi-billion-line codebase—all housed in an infamous monorepo—and employs about 30,000 software engineers to add to and maintain it. Google is known for its sophisticated processes and vulnerability management is no exception.

Luckily for us, Google has worked with O'Reilly to put together the book Software Engineering at Google (published in March 2020 and freely available in PDF) and we can learn from it.

How Google finds vulnerabilities

Google finds vulnerabilities primarily through static code analysis. Google also lays out strict criteria for vulnerability discovery and creates a healthy feedback loop so that its static code analysis can always be improving.

False positives are common in many static code analysis tools, so to maintain developer confidence, the authors of Software Engineering at Google write that the people behind Google's Tricorder static analysis platform have a "relentless focus on … deliver[ing] only valuable results to its users."

Google has four criteria for new static analysis checks:

- Be understandable,
- Be actionable and easy to fix,
- Produce less than 10% effective false positives; and
- Have the potential for significant impact on code quality.

To further hone the usefulness of Tricorder, Google has added "Not useful" and "Please fix" buttons in the UI so developers can share their feedback. This way, unhelpful checks can be removed or improved.

When it comes to actually using static code analysis, most developers will view and use static code analysis in their IDEs and during code review for only the changed portion of their code. At Google, however, the company also exposes static code analysis for the entire codebase in its code search and browsing tool.

This exposure is particularly useful at Google because it enables security teams to view all instances of a problem when they try to fix something that might be present across the entire codebase.

How Google fixes vulnerabilities

The Log4j vulnerability wasn't the first of its kind and it won't be the last. This kind of large-scale problem requires a similarly large-scale strategy, one Google deployed via their internal, large-scale code changes tool.

In Software Engineering at Google, the authors write about a vulnerability in 2016 similar to Log4j: "In early 2016, a vulnerability in the Apache Commons library allowed any Java application with a vulnerable version of the library in its transitive classpath to become susceptible to remote execution."

And in a post on Operation Rosehub, Googlers recount how their code changes tool would've helped with a similar issue: "Internally at Google, we have a tool called Rosie that allows developers to make large-scale changes to codebases owned by hundreds of different teams."

Google developers frequently apply large-scale changes to fix security issues across Google's entire codebase. But at the time, no such tool existed inside most other companies or for the open source universe. So, Google "recruited even more engineers from around Google to patch the world's code the hard way," eventually patching 2,600 open source projects.

The 2017 blog post retelling the Operation Rosehub story ends with a plea for devs to pay “attention to the fact that the tools now exist for fixing software on a massive scale“. This is one of the inspirations for Sourcegraph’s Batch Changes, which helped many organizations find and fix the recent Log4j issue across their entire codebase.

Microsoft uses an internal tool to manage open source software and dedicates a team to fixing vulnerabilities

Microsoft, like any tech company large or small, depends on third-party software, including open source projects. The risk with depending on open source software, however, is that there can be vulnerabilities introduced to your codebase via dependencies you may or may not be tracking.

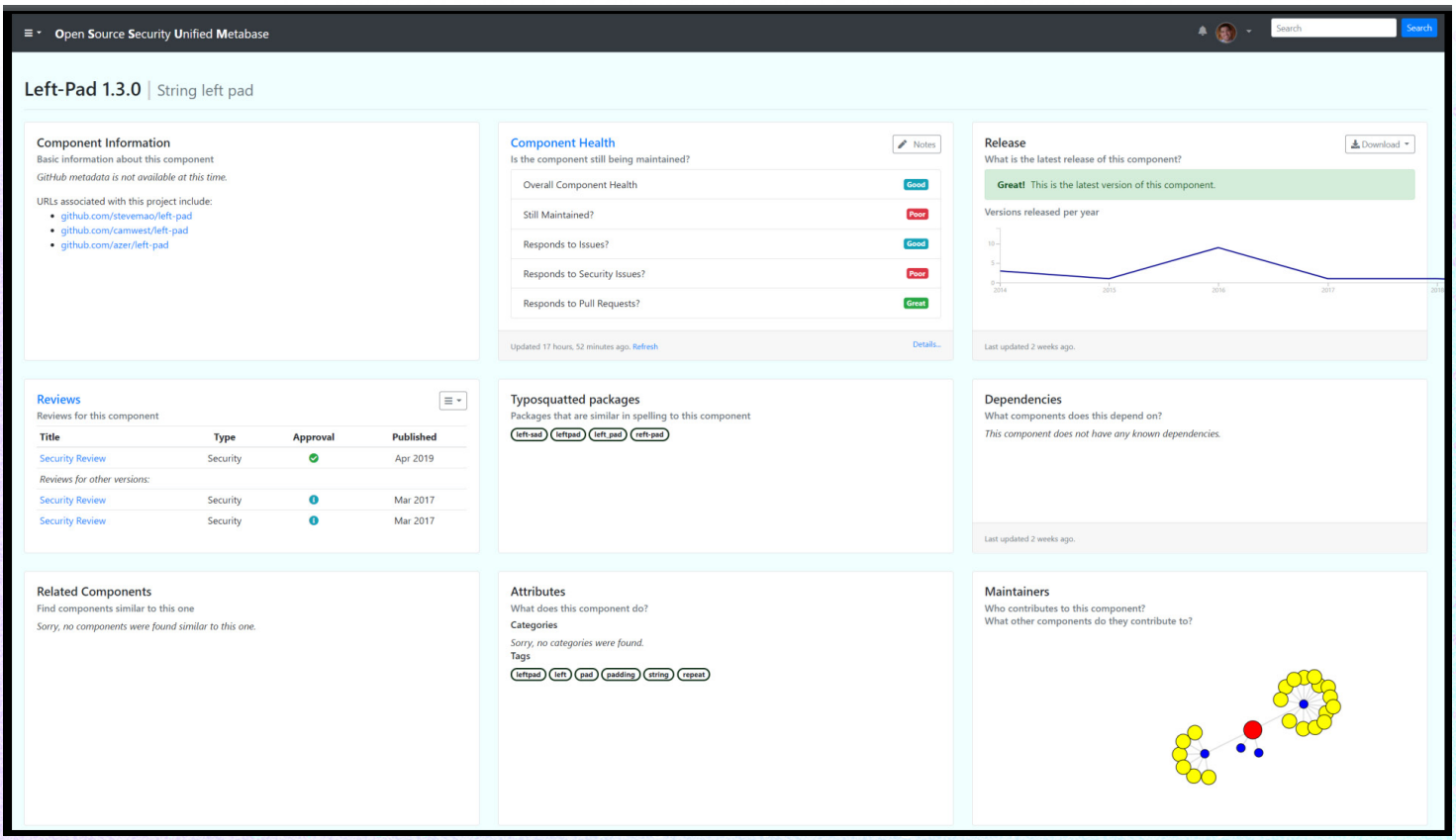
Microsoft lays out the benefits of open source as well as their best practices for using it in a page on open source security and, in a 2019 presentation at LocoMocoSec linked on that page, explains in greater depth how to securely manage open source in the enterprise.

How Microsoft finds vulnerabilities

The foundation to Microsoft’s efforts toward finding vulnerabilities is inventorying the open source software it’s using and checking to make sure that it’s secure. Microsoft adopts, and recommends other companies adopt, automated tooling that is likely already available in modern agile development environments. There are open source tools, such as NPM Audit (an NPM command which asks for a report of known vulnerabilities after sending a description of dependencies in your project to your default registry), and commercial tools, such as Snyk, which, among other features, performs software composition analysis to show you which dependencies you’re using contain vulnerable versions. Microsoft doesn’t recommend a particular tool for open source component inventory management but does recommend organizations evaluate any given tool carefully before adoption.

In an interview, Aanchal Gupta, Vice President of Azure Security, emphasizes the importance of the above effort as part of an industry-wide effort to encourage companies to create and use a software bill of materials, or SBOM. Gupta compares the current process of installing third-party software to buying food without an ingredient list. Otherwise, she argues, when a vulnerability emerges, you’re left to play “whack-a-mole” with your codebase.

Microsoft uses an internal tool called Open Source Security Unified Metabase that displays useful information about a given open source component, such as component health and maintainers.



Microsoft encourages its developers to generate open source component inventories at a "natural point" in the development lifecycle, such as during pull request validation and branch merging. Teams then store inventory results in a central place that security engineers, among others, can access.

Once Microsoft teams have assembled an inventory, developers perform a security analysis on all identified components to ensure they don't have security vulnerabilities. Microsoft breaks this effort down into four strategies:

- Check for public vulnerabilities via reported CVEs and public databases.
- Use commercial security intelligence to add more vulnerability sources.
- Perform static analysis to ensure open source components don't contain unreported vulnerabilities.
- Do security reviews on every open source component.

Microsoft, given its greater resources than most companies, has an entire team dedicated to performing security reviews, as well as support from their engineers. Microsoft notes that this is expensive, but argues that their process "offers a high assurance that components meet security requirements."

How Microsoft fixes vulnerabilities

When Microsoft teams respond to reported security vulnerabilities, they are to report those vulnerabilities to a dedicated team–the Microsoft Security Response Center (MSRC). Microsoft requires engineers to integrate open source software security response into their security programs.

When an open source vulnerability is discovered, numerous questions are asked, including:

- Are the open source maintainers trustworthy?
- Are the open source maintainers competent?
- Will the open source maintainers fix the open source component?
- Will the open source maintainers respond at all?

Finally, Microsoft asks: If a fix won't be made available by the author, should we fork or announce?

One example of the MSRC process, which Aanchal Gupta shared in the previously cited interview, involved the email-based attack from Nobelium. Gupta and her team learned of the vulnerability in November of 2021, when the MSRC began debugging it with FireEye. In December, over 500 analysts and researchers from Microsoft worked to fix the vulnerability for both the company and the wider supply chain. In line with that effort, Microsoft published over thirty blog posts in a month to share indicators of compromise and guidance for how companies could stay safe.

Outside of emergency, out-of-band fixes, Microsoft deploys patches on a regular, predictable cadence. When fixes are ready, Microsoft gathers them and distributes them on the second Tuesday of the month, known as Patch Tuesday. To make patches easier to use, the MSRC publishes bulletins on its Security Update Guide and uses Common Vulnerabilities and Exposures, or CVE, and identification numbers for each vulnerability. Bulletins also include steps for remediation and links to Knowledge Base articles with more information.

Lyft automates vulnerability scanning and reporting so engineers can focus on resolution

In a post on Lyft's engineering blog, Nicolas Flacco, software engineer at Lyft, explains that vulnerability management can be "incredibly time consuming." Typically, he explains, vulnerability management involves both scanning systems for vulnerabilities and fixing the discovered vulnerabilities. For a long time, this process was largely manual at Lyft and required scanning and tabulating results as well as deduplicating issues. When all was said and done, it could take from three months to a year to fix a discovered vulnerability.

Luckily, he explains, a lot of this manual work could be automated, and Lyft now has a process that's freed its security engineers to focus on understanding and fixing vulnerabilities.

How Lyft finds vulnerabilities

The goal of Lyft's automated vulnerability management process is to ensure human intervention is only necessary for triage and for assigning issues to service owners. Otherwise, automated tooling runs scans, converts scan results to Jira issues, and deduplicates issues.

Lyft does this via three workflows:

1. Scan one or more URLs and generate an XML report.
2. Consume the XML report and generate Jira issues for new vulnerabilities.
3. Generate the quarterly report and upload it to their document repository.

Each workflow consumes data from the previous workflow, which Nicolas diagrams:

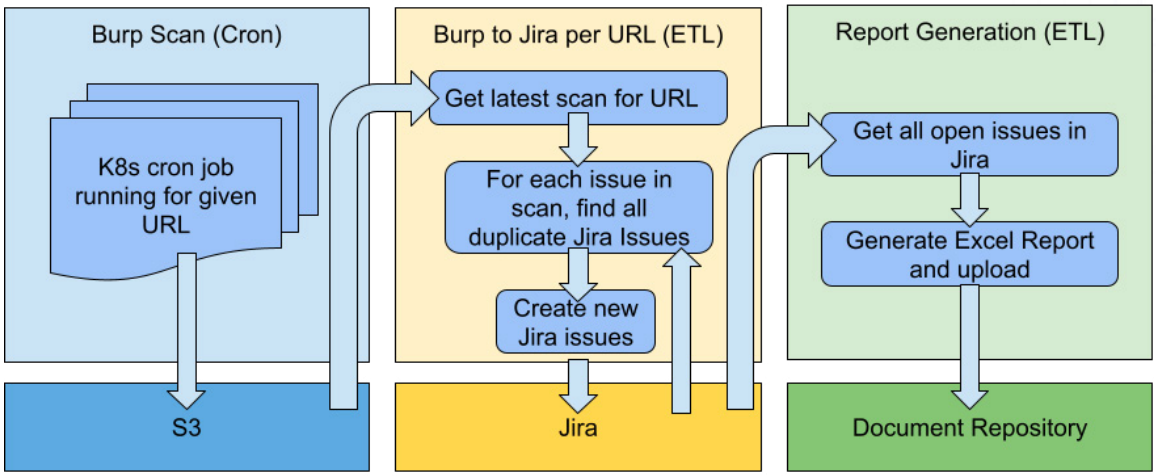


Image by Nicolas Franco via eng.lyft.com

Lyft uses cron jobs to ensure scans happen regularly. One problem, Nicolas explains, is passing the scan results to the ETL pipeline. The team stores the findings in S3 and has an ETL query S3 for the latest results, since the cron job cannot save the data to the data warehouse.

Deduplication is another potential issue. Before Lyft's automated process creates new issues in Jira, it queries existing issues to determine vulnerability type and URL. If an issue already exists, the system determines there's a duplicate and doesn't create a new issue. "This is great," Nicolas writes, "because it reduces
the amount of issues in Jira."

Lyft has also worked toward making scans more atomic by scanning a single URL and generating issues in Jira for that single URL. Doing this makes the process incremental, allowing Lyft engineers to add new URLs to the system as necessary.

The end result is that Lyft can create an entire vulnerability report via a single Jira query. Lyft assigns different resolution service level agreements, or SLAs, to different severities and since Lyft tracks when someone originally created an issue, it's easy to calculate overdue issues.

How Lyft fixes vulnerabilities

Nicolas does not go into as much detail about how Lyft fixes vulnerabilities, likely because vulnerability remediation can be quite idiosyncratic. The goal of this automation, he writes, is to ensure "our engineers are no longer occupied by trivial work, and can focus their energy on understanding vulnerabilities and driving their resolution."

Automation saves Lyft engineers three months every year, giving them extra time and space to figure out the intricacies of every vulnerability and remediate them expeditiously.

GitLab uses vulnerability feeds to find vulnerabilities and templates to help developers fix them

GitLab, like Sourcegraph, provides an open and public handbook that includes much of the information available about the processes that support GitLab's operations. In a detailed page on vulnerability management, maintained by Laurence Bierner, Director of Security Engineering, GitLab lays out its vulnerability lifecycle, backlog review, and issue review process.

How GitLab finds vulnerabilities

GitLab centers its vulnerability management process on vulnerability reports. Vulnerability reports all result in issues in GitLab but emerge from different sources, including:

- HackerOne reports imported as issues.
- Reports sent directly to GitLab.
- Issues created from security scanner results.
- Confidential reports from GitLab Team Members.

Once GitLab finds a vulnerability, a security engineer finds the team that owns the affected feature and notifies the relevant engineering and product managers. If ownership isn't clear, security engineers either use git blame to find who modified the code last or ask in Slack. The goal is to determine a directly responsible individual, or DRI.

The security engineer will set up severity and priority labels for the issue, which will determine the due date for remediation as well as the service level objective, or SLO. A section in GitLab's Security Engineering handbook page describes how they determine severity and priority and links to a CVSS calculator. For the most severe vulnerabilities, for example, GitLab requires mitigation within 24 hours, remediation on or before the next security release, and time to resolution with disabled feature flag within 30 days or before the feature flag is enabled.

How GitLab fixes vulnerabilities

GitLab follows a strict security release process to fix vulnerabilities. Engineers either remediate vulnerabilities as part of the regular monthly release or, in the case of critical vulnerabilities, remediate them in a critical security release.

GitLab reinforces the idea that developers should be verifying whether an issue is truly a vulnerability. Only after doing so should they proceed with the following six steps:

1. Use a security issue template to create a security implementation issue on the relevant security repository.

2. Perform "additional solution evaluation" if the given vulnerability has "far-reaching impact" or is a "breaking change."

3. Submit a security merge request that targets the default branch with a security fix that uses the security merge request template.

4. Prepare the backports only after the merge request has been approved by an AppSec team member.

5. Approve the backports. Only the same maintainer that reviewed and approved the merge request can do this approval.

6. Assign the merge request and the backports to the @gitlab-release-tools-bot.

As you can see, the process uses numerous templates and codified steps, and there is even more detail in sections of the vulnerability management page that further describe the security implementation issue, security merge requests, and backports.

GitLab takes care to describe what developers should do and what they should not do. According to this page, a GitLab developer's "main concern" is not revealing the vulnerability before the company is ready to disclose it. As such, GitLab developers are told not to push to gitlab-org/gitlab but are instead to deploy fixes in particular, separate repos.

Atlassian uses manual and automatic methods to find and fix vulnerabilities

In a paper it produced about its approach to vulnerability management, Atlassian recognizes, up front, that security vulnerabilities are an "inherent part of any software development process."

The goal of their vulnerability management process is to "reduce both the severity of and frequency with which vulnerabilities arise in our own products and services." To do so, they combine automated and manual processes, both for finding vulnerabilities and fixing them.

How Atlassian finds vulnerabilities

Like Microsoft, Atlassian lays its vulnerability management system on a foundation of internal and external network asset discovery. They use a home-grown system to inventory EC2 and Load Balancer AWS Assets, which itself uses AWSConfig, and notes the correct owners of each asset. Atlassian retains as many as 60 million assets per year as part of this process.

The actual discovery of security vulnerabilities involves a range of methods and tools to automatically scan for vulnerabilities, including:

- Host-based scans: Atlassian uses Assetnote for continuous security scans of its external perimeter and Nexpose for continuous internal and external scans.
- Container image scans: Atlassian uses Snyk to scan container images when Atlassian deploys them into production or pre-production environments.
- Open source dependency scans: Atlassian uses Snyk here, too, to find vulnerabilities that exist in open source dependencies.
- AWS configuration monitoring: Atlassian uses Trend Micro Cloud One - Conformity to continuously monitor their configuration against established baselines.
- A bug bounty program: Atlassian uses Bugcrowd to run their bug bounty program. Bugcrowd, according to Atlassian, provides the company with "access to an expert, trusted community" who can test Atlassian's products and report on any discovered vulnerabilities.
- Customer and user reports: Atlassian enables its users to report bugs they might find to the Atlassian support team.
- External penetration testing: Atlassian uses "specialist security consulting firms" to conduct penetration tests on infrastructure as well as on projects Atlassian deems "high risk."
- Product security reviews: Atlassian has a product security team that completes "targeted code reviews."
- Red team: Atlassian has a Red team, an internal team that simulates what adversaries might do to exploit vulnerabilities, to find even more vulnerabilities.

Atlassian emphasizes that both its tools and its processes for finding vulnerabilities are under "continuous" review and iteration.

How Atlassian fixes vulnerabilities

Atlassian initiates and tracks the vulnerability fixing process via a ticketing and escalation system that assigns a ticket to each vulnerability and then assigns each ticket to a relevant product team.

Atlassian also maintains a public Security Bugfix Policy, which contains varying levels of SLOs for different vulnerabilities. Critical severity bugs, for example, require fixes in product within two weeks of being reported; low severity bugs, on the other hand, require fixes within 25 weeks.

Additionally, Atlassian staffs a security team that oversees the vulnerability remediation process. This team works with the product and infrastructure teams to help them figure out and remediate vulnerabilities.

Key to this process is testing and deployment. Atlassian reports that it tests every fix "thoroughly" and either incorporates the fix into its CI/CD pipeline for its cloud products or rolls the fix into a new release for its server and data center products. Atlassian then closes the loop when security scans show that a given vulnerability does not reappear or, if a vulnerability was found manually, then a product, infrastructure, or security team member confirms a fix is available to customers.

Learn from the best and handle vulnerabilities like the best

The above examples show that there are endless opportunities for innovation in vulnerability management. By learning from the best companies, you too can find and fix vulnerabilities efficiently and effectively, enabling your teams to maintain a secure codebase and report confidence in that security to your customers and partners.

Here's a bonus example: Nutanix, a hybrid multi-cloud provider with 20,000 customers, an annual revenue of nearly $1.394 billion, and over 6,000 employees uses Sourcegraph to find and fix security vulnerabilities.

When Log4j emerged in December 2021, Nutanix was able to:

- Find JMSAppender–an indicator that a particular, vulnerable version, Log4j 1.x, existed–fix it, and send out a release in less than 5 minutes.
- Deliver patches to its customers that fully remediated the Log4j vulnerability in under 4 days.
- Identify every instance of Log4j across its sprawling codebase with 100% confidence.

"Sourcegraph was the right product at the right time," reports Jon Kohler, Technical Director of Solution Engineering at Nutanix.

The Sourcegraph code intelligence platform provides numerous ways to find and fix vulnerabilities, including code search, which can immediately discover vulnerabilities across all of your repositories; Code Insights, which can track and visualize the progress of deployed fixes; and code monitoring, which can alert you when insecure code is added to your codebase.

To learn more about how Sourcegraph can help you with code security, check out our page on finding and fixing vulnerabilities or request a demo.

about.sourcegraph.com

Sourcegraph is a code intelligence platform that unlocks developer velocity by helping engineering teams understand, fix, and automate across their entire codebase.