Security for open-source maintainers

Maya Kaczorowski
Sr Director Product Manager, GitHub
@mayakacz @MayaKaczorowski

Maya Kaczorowski

Product Manager, Software Supply Chain Security

GitHub

Agenda

Security issues in open source

What your project can do

- Vulnerability management
- Security automation
- Auditing

What you can do as a maintainer

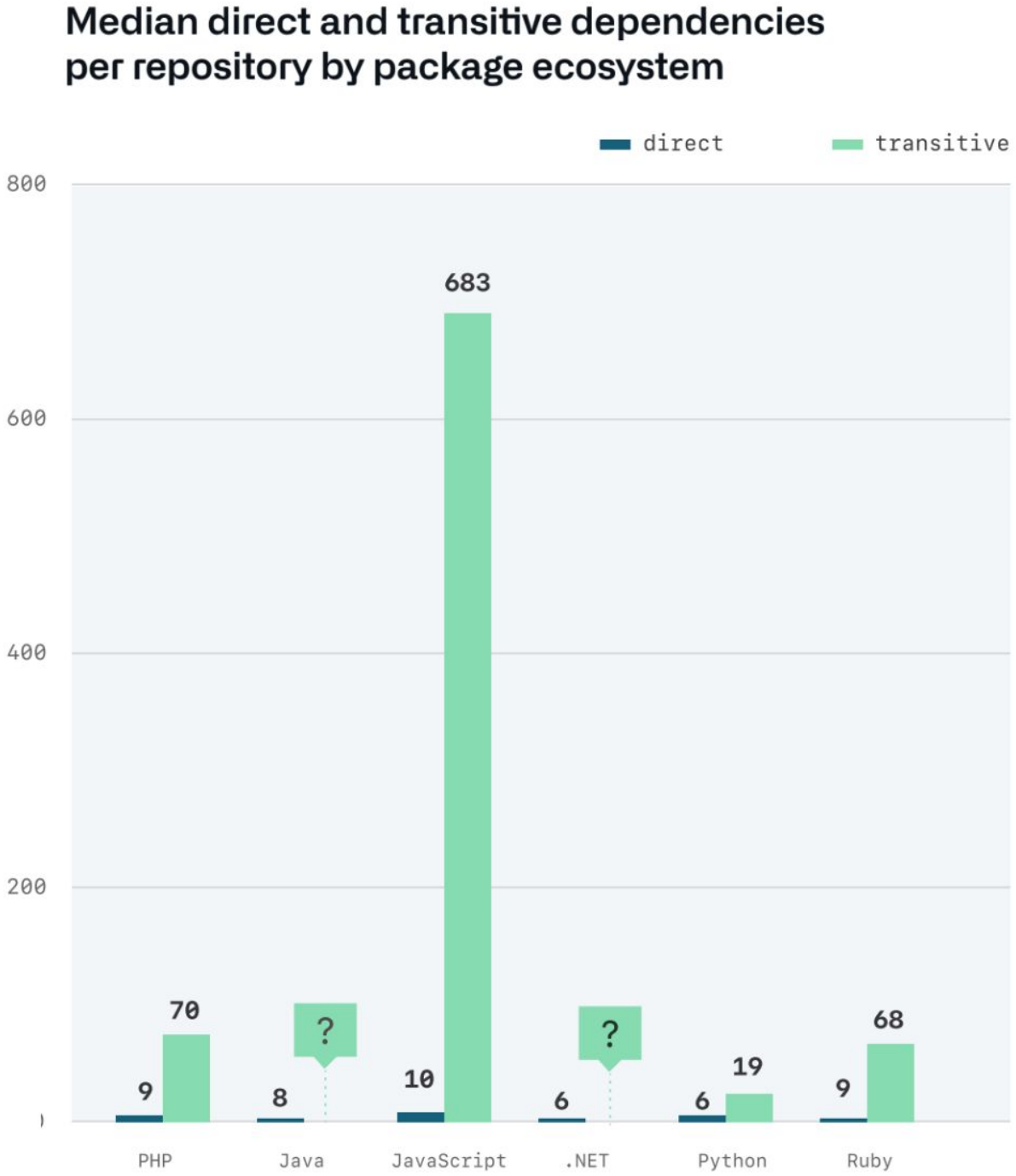
- Keep your account safe

What's different for commercial open source

Security issues in open source

Vulnerabilities move as quickly as great ideas, or viruses

@MayaKaczorowski



Software dependencies are pervasive

The impact of an open source security attack can be much greater - affecting all its dependents

Supply chain attacks

Method

- Malicious code, e.g., backdoor, malware, known vulnerability
- Compromised build tool
- Compromised signing keys
- Compromised package manager
- Compromised vulnerability reporting
- Account takeover
- Project takeover
- Accidental, e.g., typosquatting
- Deletion

Goal

- Backdoor, e.g., targeted, watering hole
- Malware, e.g., cryptocurrency mining
- Service disruption, e.g., deletion

event-stream

Widely used npm library
CVE-2018-1000851

Method: Project takeover

“he emailed me and said he wanted to maintain the module, so I gave it to him. I don't get any thing from maintaining this module, and I don't even use it anymore, and havn't for years.”

Goal: Backdoor

Highly targeted to Copay developers, to distribute malware to capture credentials for Dash Copay bitcoin wallets

- `right9ctrl` volunteered to take over the package on GitHub, then added `flatmap-stream` as a dependency
- Another user `hugeglass` added malware to steal credentials to bitcoin wallets to `flatmap-stream`
- Profit

https://blog.npmjs.org/post/180565383195/details-about-the-event-stream-incident

eslint

npm package for Javascript static code analyzer

Method: Account takeover

Credential stuffing

Goal: Backdoor

To get `.npmrc` npm publishing creds!

- Attacker generated new auth tokens and published two malicious packages
- Second package discovered within an hour; altogether published for <4 hrs
- npm revoked ALL access tokens before the incident
- “A very small number” of packages and users affected

https://eslint.org/blog/2018/07/postmortem-for-malicious-package-publishes
https://gist.github.com/hzoo/51cb84afdc50b14bffa6c6dc49826b3e
https://status.npmjs.org/incidents/dn7c1fgrr7ng

docker123321

17 Docker Hub images

Method: Accidental

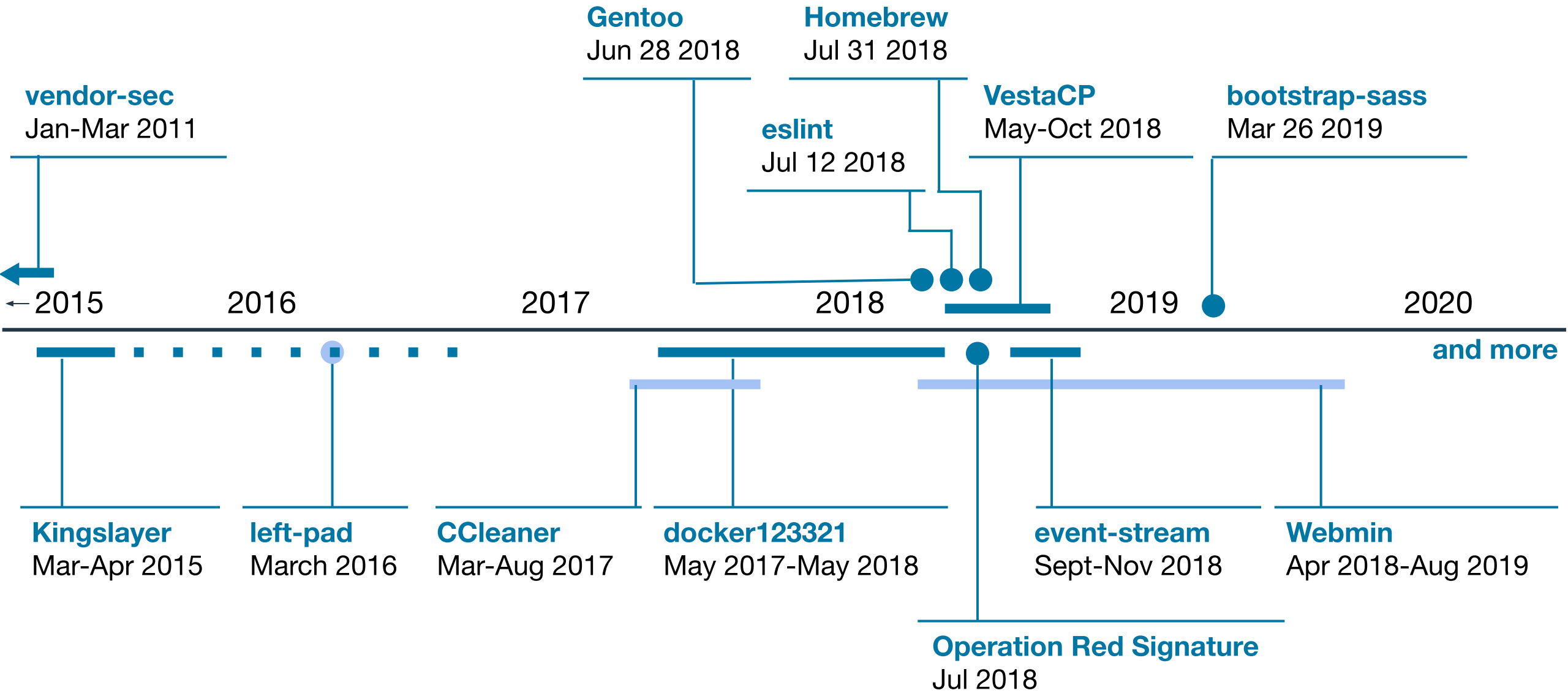
Easy to type registry name

Goal: Malware

Mining ~$90k of Monero

- 17 images in `docker123321` registry
- Contained malware since at least July 2017
- Suspected malware, positively identified as part of a cryptomining botnet
- Removed by Docker Hub in May 2018

https://kromtech.com/blog/security-center/cryptojacking-invades-cloud-how-modern-containerization-trend-is-exploited-by-attackers

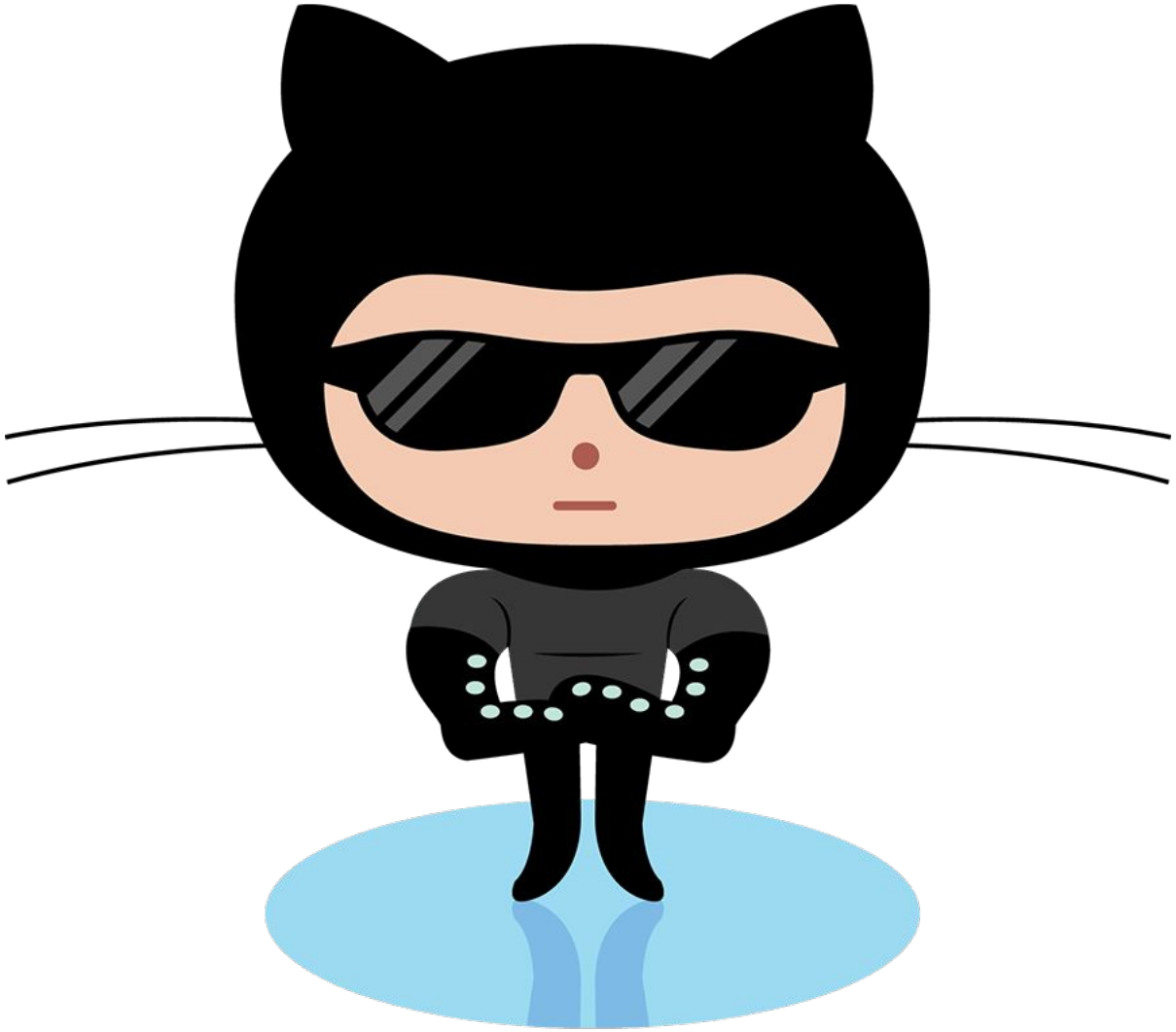


Timeline of select (known) Software Supply Chain attacks

What your project can do

What your project can do

- Vulnerability management
- Security automation
- Auditing



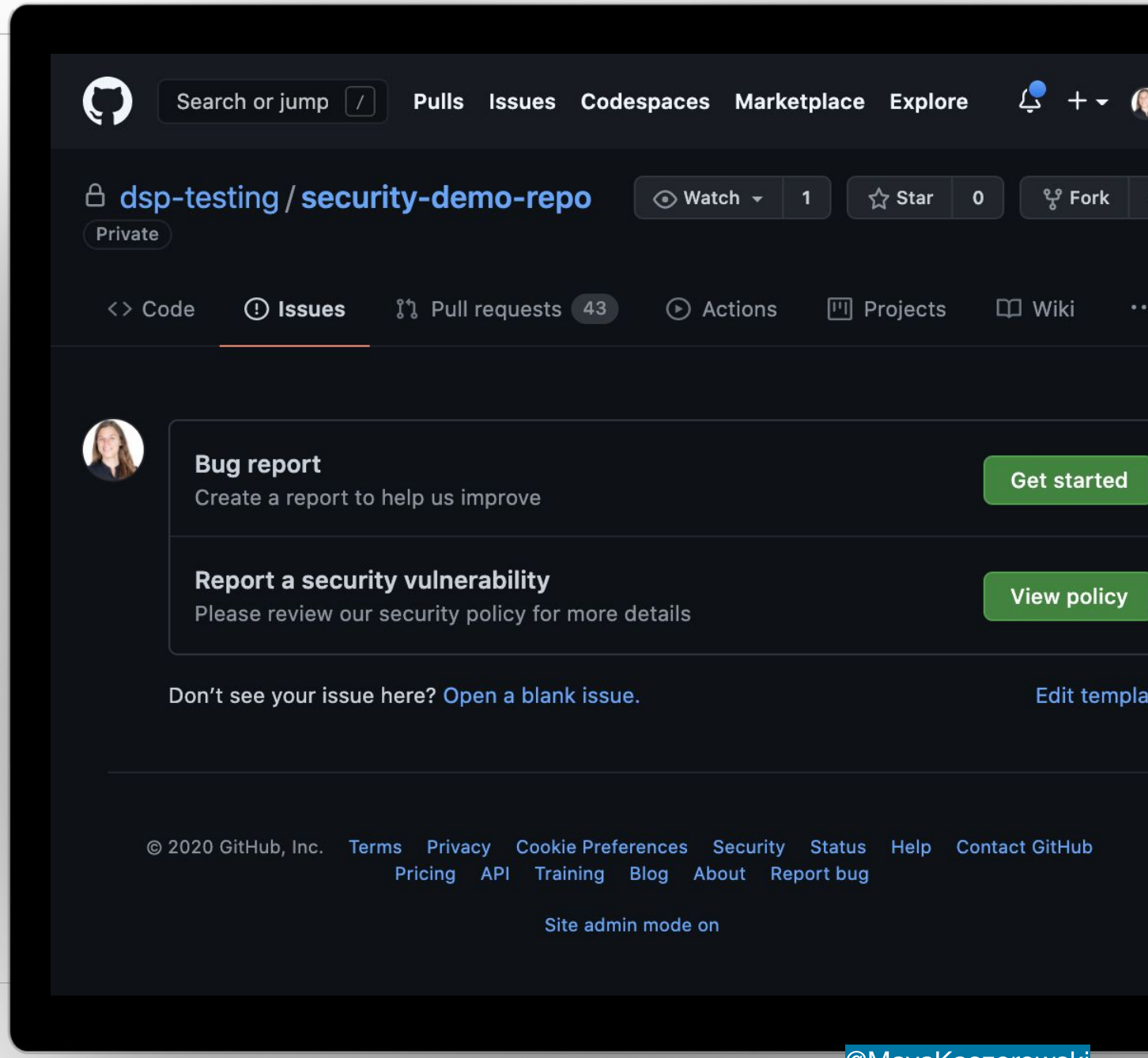
Vulnerability management

Fix and publish vulnerability information

- **Respond** to a report of a security issue in your project
- **Develop the fix**, addressing the vulnerability, ideally before it's public
- **Release the fix** and backport to any supported versions
- **Notify your users** that a patch addresses a security vulnerability

Security policy: SECURITY.md

1. How you want to be contacted
2. What versions you support



Security policy: What you might want to ask for

Details on a vulnerability:

- Affected versions and vulnerable component(s)
- Reproducible steps
- Logs, if applicable
- Impact

Security policy: Safe Harbor

Lets a researcher know you won't pursue legal action for finding security issues

"No lasting damage"

- Privacy violations
- Destruction of data
- Service interruption
- Social engineering

Security policy: Tools and examples

Build a policy:

https://hackerone.com/policy-builder/

https://securitytxt.org/

Examples:

https://github.com/microsoft/microsoft.github.io/blob/master/SECURITY.MD

https://github.com/electron/electron/blob/master/SECURITY.md

https://github.com/goharbor/harbor/blob/master/SECURITY.md

Fix a vulnerability: Develop and release a fix

Respond to the researcher and acknowledge the report

Validate the issue

- Ask clarifying questions
- Update them on your progress

Develop a fix privately (if possible)

- Bring in developers who have expertise and capacity to help fix

Fix a vulnerability: Obtaining a CVE

CVE = Common Vulnerability & Enumeration

What's affected: package(s)/component(s) and version(s)

Severity: Critical, High, Moderate, Low

Description of the issue

Published at https://cve.mitre.org/cve/

Fix a vulnerability: Notify users

Once you have a fix, tell your users to patch!

Explain it's a security issue

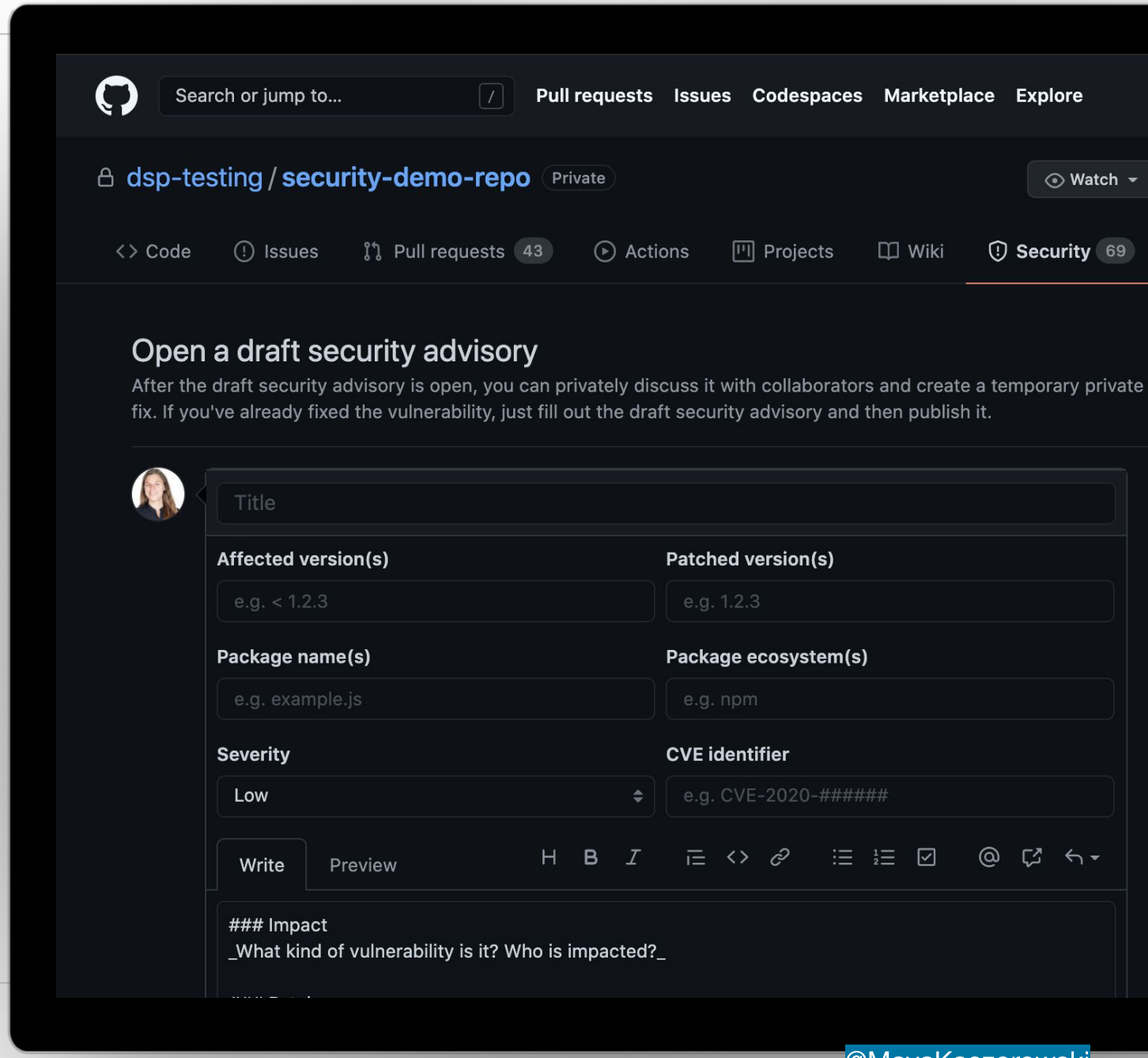
Part of your normal release process or out of band

Fix a vulnerability: Security Advisory

Collaborate on a fix as part of a private fork

Request a CVE

Publish when you're ready to go public



Fix a vulnerability: What if it's already public?

Sometimes you can't control when a vulnerability is disclosed

Be transparent and honest with your community as you resolve the issue

Ask for help!

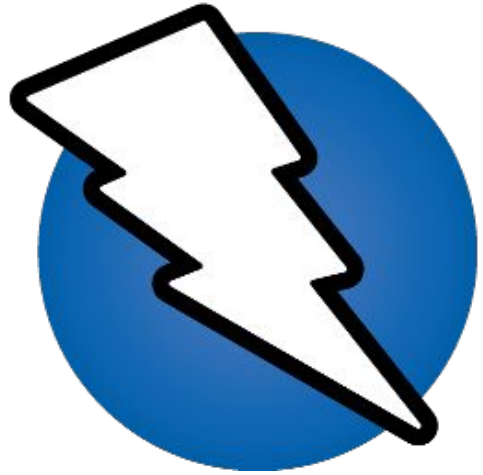
Test out your process!

Security automation: Scan for vulns

Scan for existing vulnerabilities in your code

Lots of tools are free for open source:
https://owasp.org/www-community/Free_for_Open_Source_Application_Security_Tools

e.g., SAST: CodeQL, DAST: OWASP ZAP

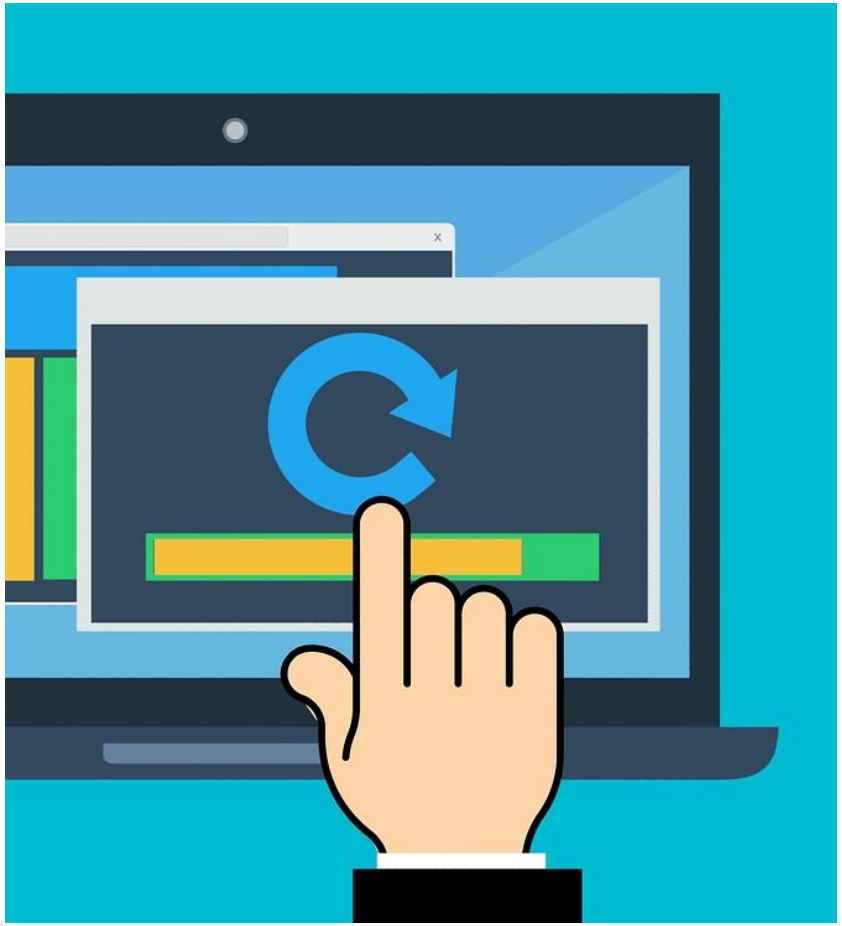


Security automation: Update dependencies

Update your dependencies, especially if they have vulnerabilities

Apply patches for known security vulnerabilities

Regularly update dependencies to get the latest versions



Security automation: Prevent vulns

“Shift left”

Catch issues earlier:

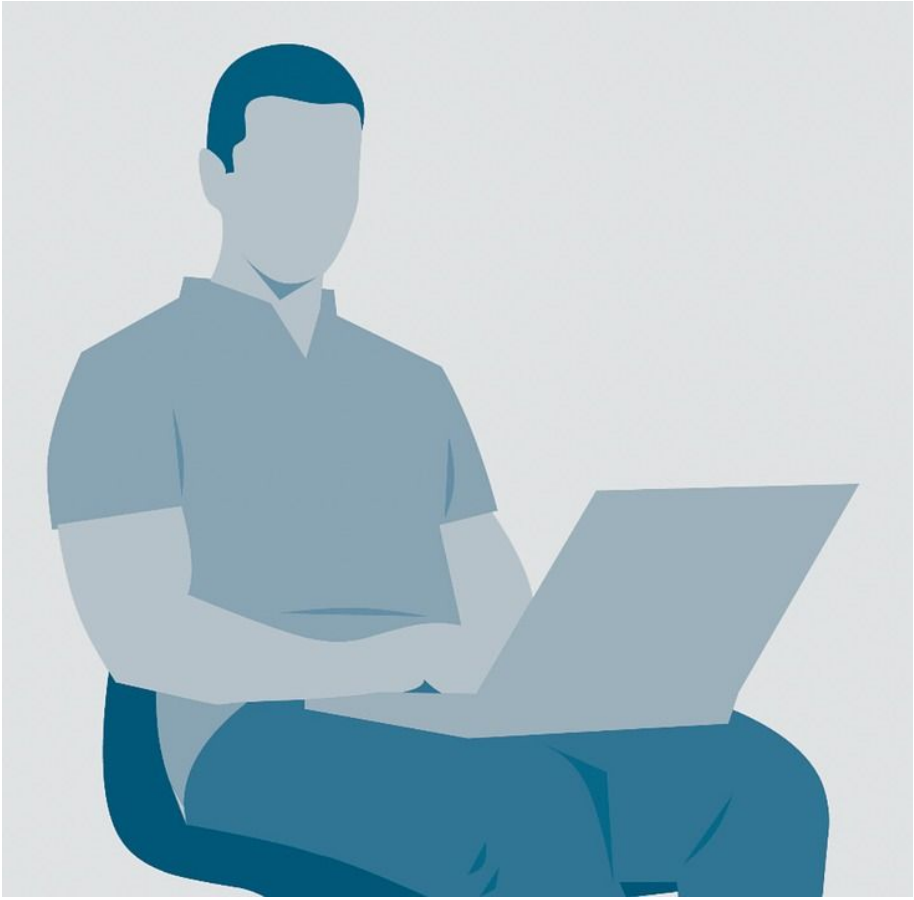
- In IDE
- In PR
- At build time
- At deployment time
- … anytime before it’s exploited!

Security automation: Enforce best practices

Implement controls to enforce best practices

- Tests and check suites
- Fail builds with issues
- Block non-compliant deployments
- ... whatever best practice you need!

Warn, then enforce



Auditing: If you can, conduct an audit

Conduct a full security review

Check data inputs/ outputs, validation

What you can do as a maintainer

What you can do as a maintainer

- Keep your accounts safe
- Make security a priority

Keep your accounts safe

Enable 2FA

Keep SSH keys safe

Don’t commit secrets in code

What’s different for commercial open source

The biggest difference between non-commercial and commercial open source?

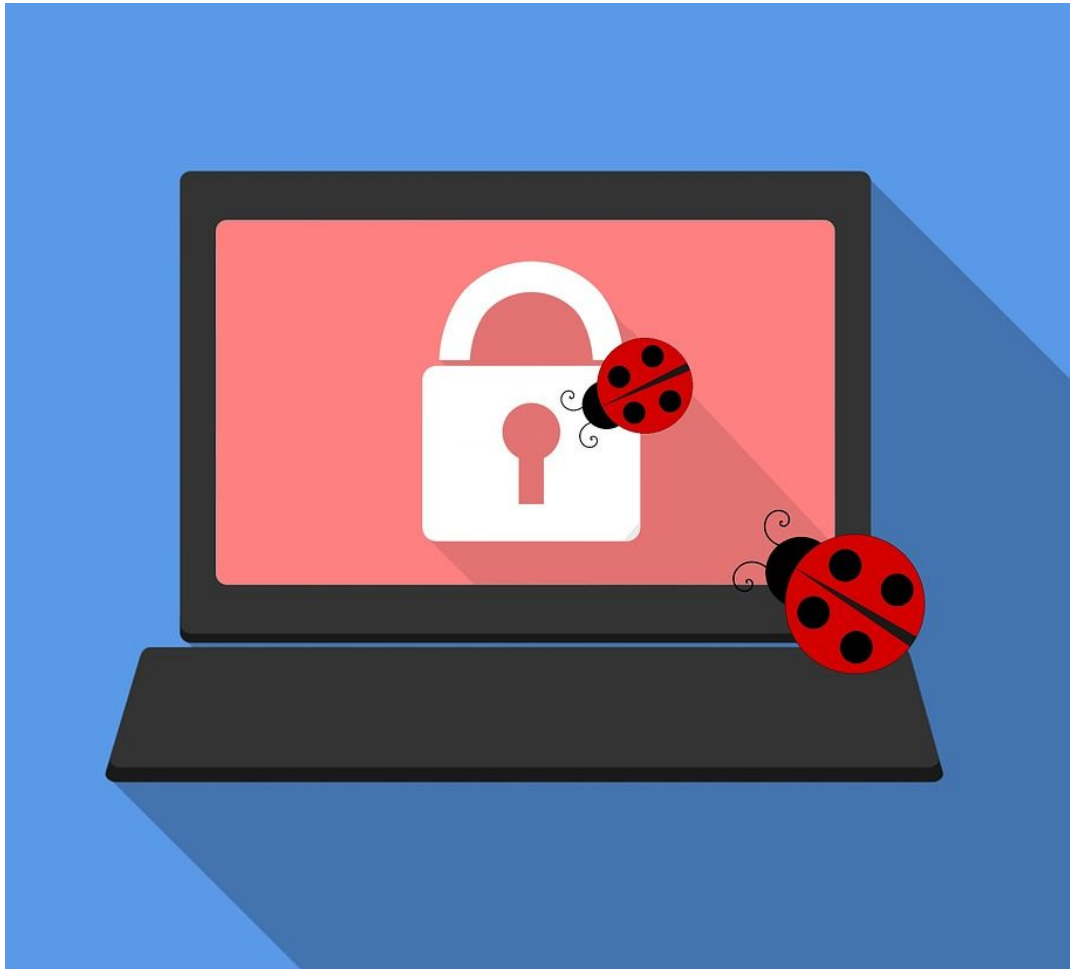
Pay your security team

What about bug bounties?

Ask yourself:

Are you able to handle an influx in reports?

If you're a commercial open source project, remember that finding bugs is hard and should be paid work



Summary

There are supply chain attacks

Your project needs a way to handle vulnerabilities

Your project can use automation to help avoid security issues

Protect your accounts with 2FA

Pay for security in a commercial project

Thank You