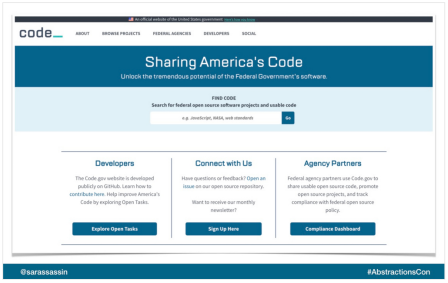
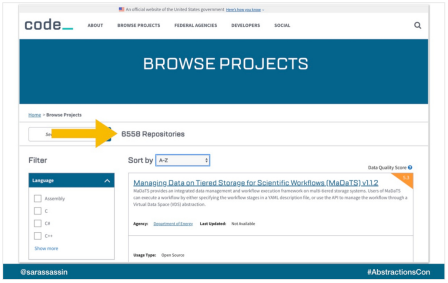
Practical Open Source Security

Sara Cope

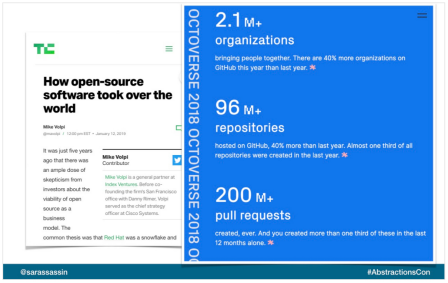


I work on an open source project that's about other open source projects and consumes open source projects so I'm basically in open source and advocating for open source all day long. It's amazing and I love it.



When code.gov was first launch in 2016 it had under 50 projects.

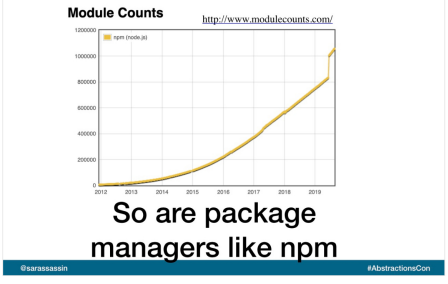
Today there are more than 6 thousand.



Popularity of open source isn't just skyrocketing in the federal government.

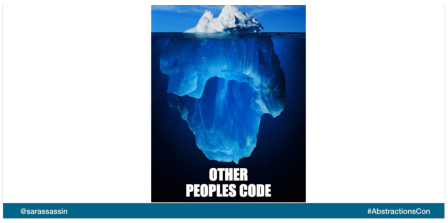
Everyone seems to either want to publish an open source project, be a maintainer or contribute to an open source project.

Which is great because that creates even more software that's available for everyone else to consume.



And we can see that in the way that package managers like NPM are growing as well. And as a result, we're adding more and more dependencies to our applications.

So we see apps shifting to use more and more dependencies so much that amount of code we're writing at times can me very small compared to the amount of code we're consuming.



Using someone else's code isn't just set it and forget, that code at any time could potentially contain vulnerabilities.

So who's responsibility is it to be sure the dependencies in your project stay secure?

caption: group of people sitting around a table playing the "nose goes" (not it) game.

This was me until the government shutdown last year. With 2 front-end devs on our team, I worked more on the design UI side while the over dev worked more on the "back of the front" stack like JavaScript, components and the like. But during the shutdown, both of the other 2 developers left our team. So I went from being on an engineering team to BEING the engineering team. I had to figure out a lot of things really quickly and security has been one of those topics. And that's really what I'm going to talk to you about today, all these security things we should be doing.

"As a developer consuming open source dependencies, it's my responsibility to make sure my dependencies are secure."

If you're using open source (and you most certainly are), you need to take on the responsibility of keeping it secure. You must set up the tools and processes that will help you stay safe, and raise awareness of this risk throughout your organization.

The 2017 Equifax breach serves as a very painful lesson of what could happen if you don't adopt a security mindset and take action to keep your application dependencies secure.

Equifax had a massive data breach that exposed the personal information of millions people. It was completely preventable. This breach was caused by an open source package dependency which had an identified vulnerability that went unpatched for 2 months.

https://www.synopsys.com/blogs/software-security/equifax-apache-struts-vulnerability-cve-2017-5638/

So I want to go over specific things you could be doing for the applications that you develop which use open source dependencies.

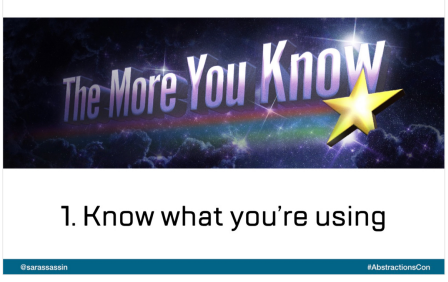
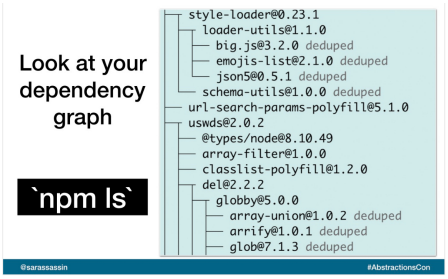


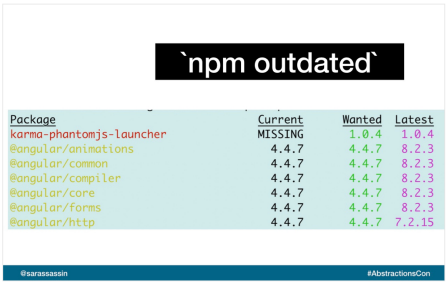
Figure out the dependencies you have and what packages those depend on.

Not everyone comes in to a project at the beginning and knows exactly what's being used. So take the time to go through the dependencies list to see what's there.



Just stop and look at your dependencies. Familiarize and refresh yourself with what's there. That way when you hear about a new vulnerability you might have a better reaction time when you know that your app is using certain packages.

If you're an nom user, you can execute `npm ls` to display a dependency tree.



Use `npm outdated` to check the registry to see if any installed packages are currently outdated

The more outdated a package is, the more likely it is to itself have vulnerable dependencies.

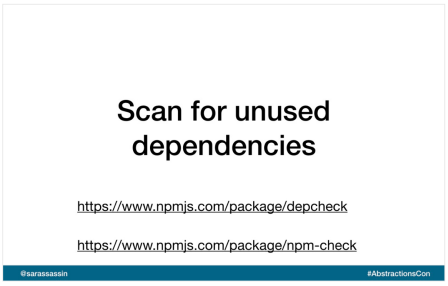
So npm outdated can tell you exactly how far behind you are in dependency versions.

It's better to patch these up on your own terms instead of waiting for a vulnerability to be disclosed that you're forced to immediately act on.

Get rid of dependencies you don't need. As the number of dependencies increase, the risk of ending up with a vulnerable package can also increase.

With the passage of time and changes in your code, it is likely that you'll stop using some packages altogether and instead add in new ones. However, developers tend not to remove old packages as they go along.

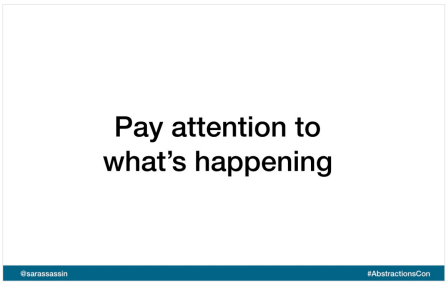
Take the time to remove unused packages.



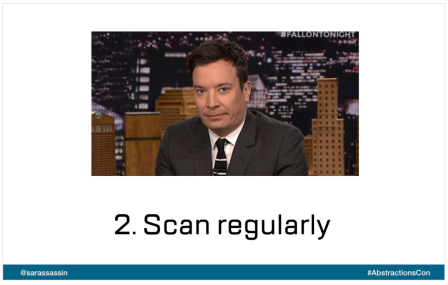
There are tools available to help with finding unused dependencies, here are a few.

https://www.npmjs.com/package/depcheck

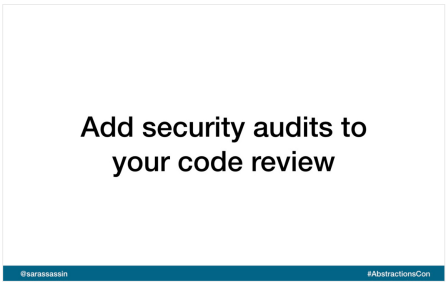
https://www.npmjs.com/package/npm-check



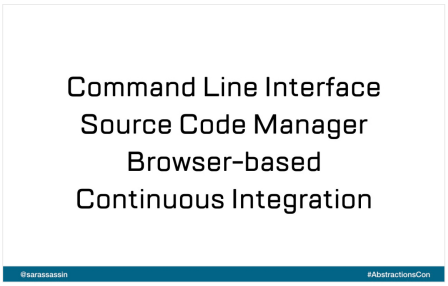
Set up monitoring mechanisms to stay aware of new vulnerabilities as they are discovered so you can keep track of security announcements affecting those dependencies and any versions released.



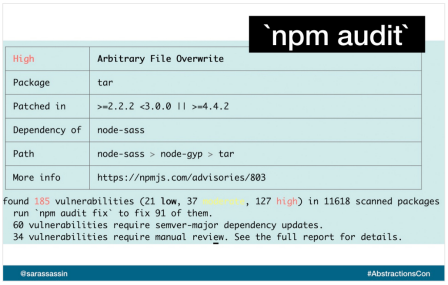
Next you want to regularly scan for vulnerabilities so you know what's going on with your dependencies and when to patch.



A security audit could exist as part of a code review where peers ensure that secure code best practices are followed, or by running different variations of security audits such as static or dynamic application security testing. Whether manual or automatic audits, they are all a vital part of detecting and reducing vulnerabilities in your application, and should be executed as regularly and early in the development phase as possible in order to reduce risks of exposure and data breaches at a later stage



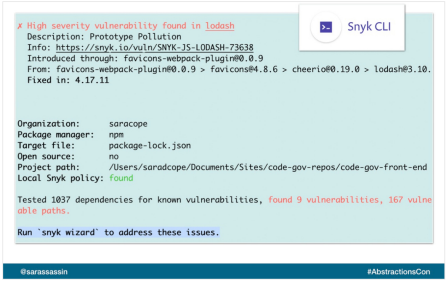
There are several approaches you can take to scan for vulnerabilities and I would recommend a combination of all of these.



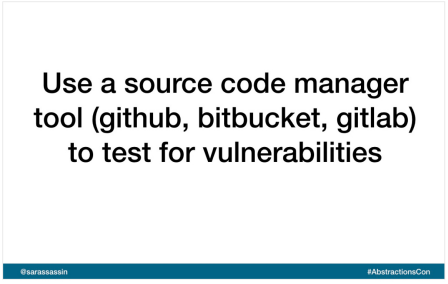
CLI:

The output varies by tool, but most will list either the known vulnerabilities or vulnerable paths, and provide information for each of them. This typically includes a title, description, severity score, and more.

It even provides `npm audit fix` to actually go ahead an upgrade those dependencies to patched releases where possible.

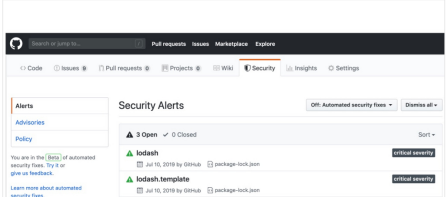


We use another tool called Snyk at the command line which can act pretty similar to `npm audit` and also provides an upgrade wizard.

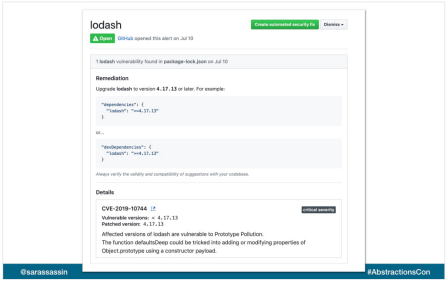


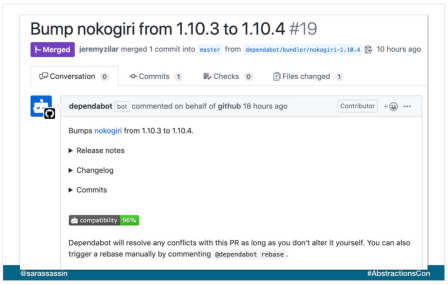
Using an SCM integration offers a few advantages over the CLI:

- Easier to test or browse multiple issues at once
- The web interface is naturally richer than the terminal, often allowing better usability
- It simplifies continuous testing and fixing, as we'll see later on

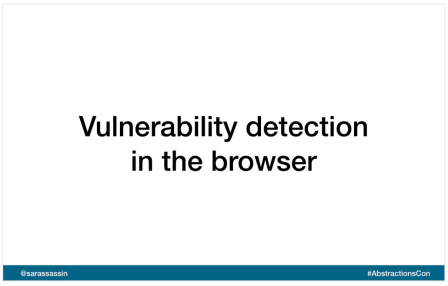


Github actually has a security tab on their interface now where you can see details of any detected vulnerabilities.

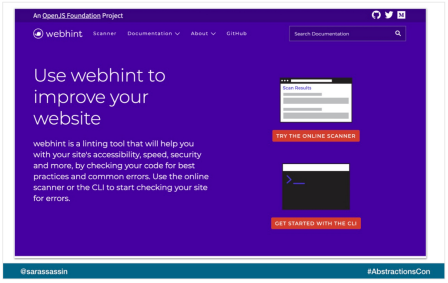




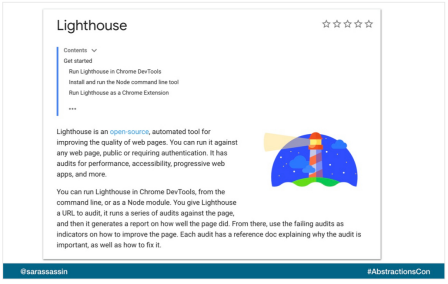
They've also added automated security fixes which you can turn on to have their dependabot submit PR's to update packages that have identified vulnerabilities.



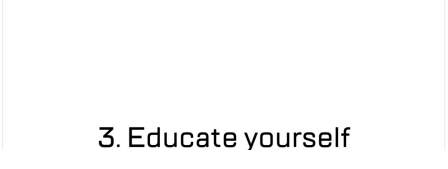
More recently, though, Google and Microsoft integrated detection of vulnerable JS libraries into their web development tools, Lighthouse and Webhint. Lighthouse is also embedded in Chrome's and Opera's browser dev tools, further simplifying and raising the visibility of this test. I encourage you to try these tools out.



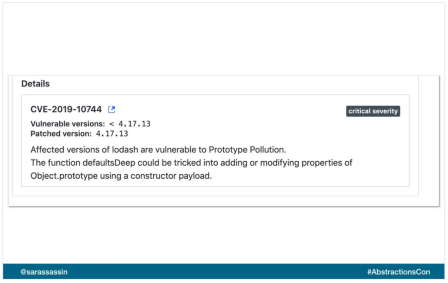
https://webhint.io/scanner



Google lighthouse



As you're seeing vulnerabilities in packages you'll hopefully want to educate yourself a bit on what exactly the issue is.



A good first place to start is to read the CVE for the vulnerability.

A CVE is created with a unique number when a specific vulnerability is found in a package.

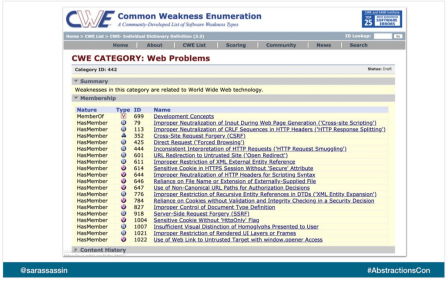
cve.mitre.org

MITRE is a government-funded organization that puts out standards to be used by the information security community.

To follow new CVE announcements, you can checkout @CVEnew on Twitter.

Mitre also keeps a list of common issues they see, or CWE's. So you can visit this database to get a better idea of specific types of security vulnerabilities.

cwe.mitre.org



There's a whole informational database about different types of vulnerabilities which is interesting and fun to dive-in to.

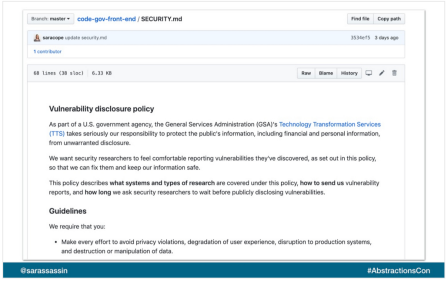
Just to recap these:

CWE stands for Common Weakness Enumeration, and has to do with the vulnerability—not the instance within a product or system.

CVE stands for Common Vulnerabilities and Exposures, and has to do with the specific instance within a product or system—not the underlying flaw.

Here's a great blog post that recaps these and more: https://danielmiessler.com/blog/mitre-quick-reference/

Establish a policy and process to quickly roll out a security fix release of your software product once supporting frameworks or libraries needs to be updated for security reasons. Best is to think in terms of hours or a few days, not weeks or months. Most breaches we become aware of are caused by failure to update software components that are known to be vulnerable for months or even years.

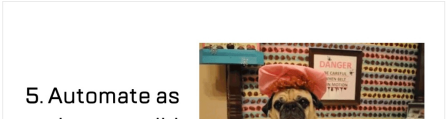


On code.gov, we've added a SECURITY.md file which give info on several areas. Here are some ideas for your security policy:

How quickly should we be releasing security fixes

How can someone let us know if we have a security vulnerability

Who to contact for more information



Continuously tracking your application's dependencies for vulnerabilities and efficiently addressing them is no simple feat. When you can, it's great to automate a lot of these things to remove some of the security burden.

much as possible

@sarassassin #AbstractionsCon

Scan on commit

https://18f.gsa.gov/2017/09/26/automated-scanning-for-sensitive-information/

@sarassassin #AbstractionsCon

Often when developing open source software, and especially software that relies on outside services, you'll find that you have to manage sensitive information. While there are a large number of things that can be considered sensitive, open source developers often deal with sensitive items such as API tokens, passwords, and private keys that are required for the system to function. One thing you can do is to scan your code on commit to check for these types of items.

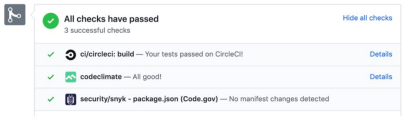
Here's a blog post which details the tool we use and includes a quick setup script: https://18f.gsa.gov/2017/09/26/automated-scanning-for-sensitive-information/

Test locally

@sarassassin #AbstractionsCon

You can add these CLI scanning applications to your tests so you can run them locally as part of your tests along with linting, accessibility testing, unit testing and anything else you might want to verify before making a pull request.

Add PR checks



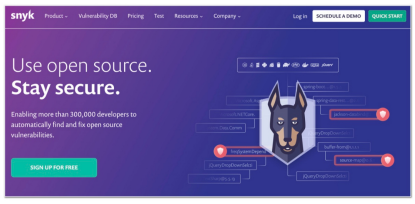
@sarassassin #AbstractionsCon

We've also added checks to our open source project using web hooks so that all pull requests are scanned for new vulnerabilities.

Add continuous monitoring

@sarassassin #AbstractionsCon

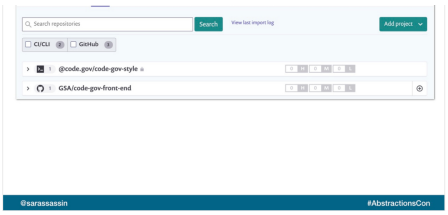
To continuously avoid known vulnerabilities in your dependencies, integrate a scanning tool into your build system.

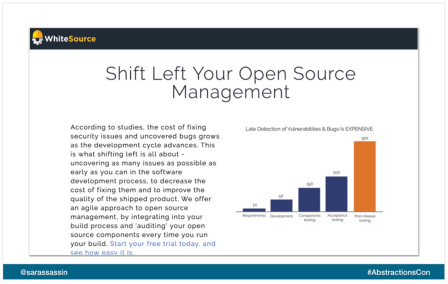


@sarassassin #AbstractionsCon

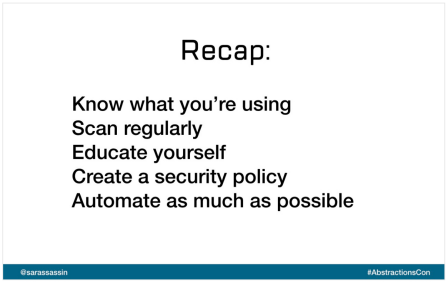
Snyk is one the tools that we use to continuously monitor our projects. It will give you an immediate notification of any security vulnerabilities as well as send you a weekly email summary of new vulnerabilities.

It actually gives you a dashboard to you can see all your projects in one place which is really helpful.

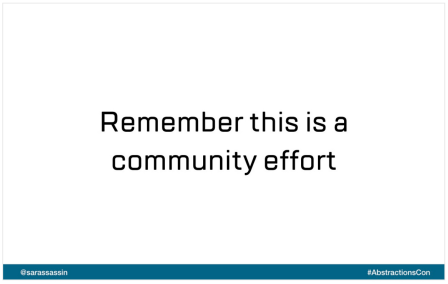




WhiteSource is another example of a monitoring tool you can use and it's actually what GitHub uses to do their security monitoring.



Just to recap these things:



And finally please remember that Open source is created by the community, and we as participants in that should work to keep it secure.

Please try to be a responsible member of both this ecosystem and your organization and accept ownership for controlling the risk from the libraries you use.

I hope you all stay secure!