SEAGL 11/09/2018

MICHAELA (MIKI) DEMETER

CHOOSING MORE SECURE OPEN SOURCE PACKAGES: LESSONS FROM THE REAL WORLD

only by examining the components in detail will you know what is inside your projects

AS A SECURITY PROFESSIONAL YOU ARE EXPECTED TO KNOW WHAT IS IN YOUR PRODUCT

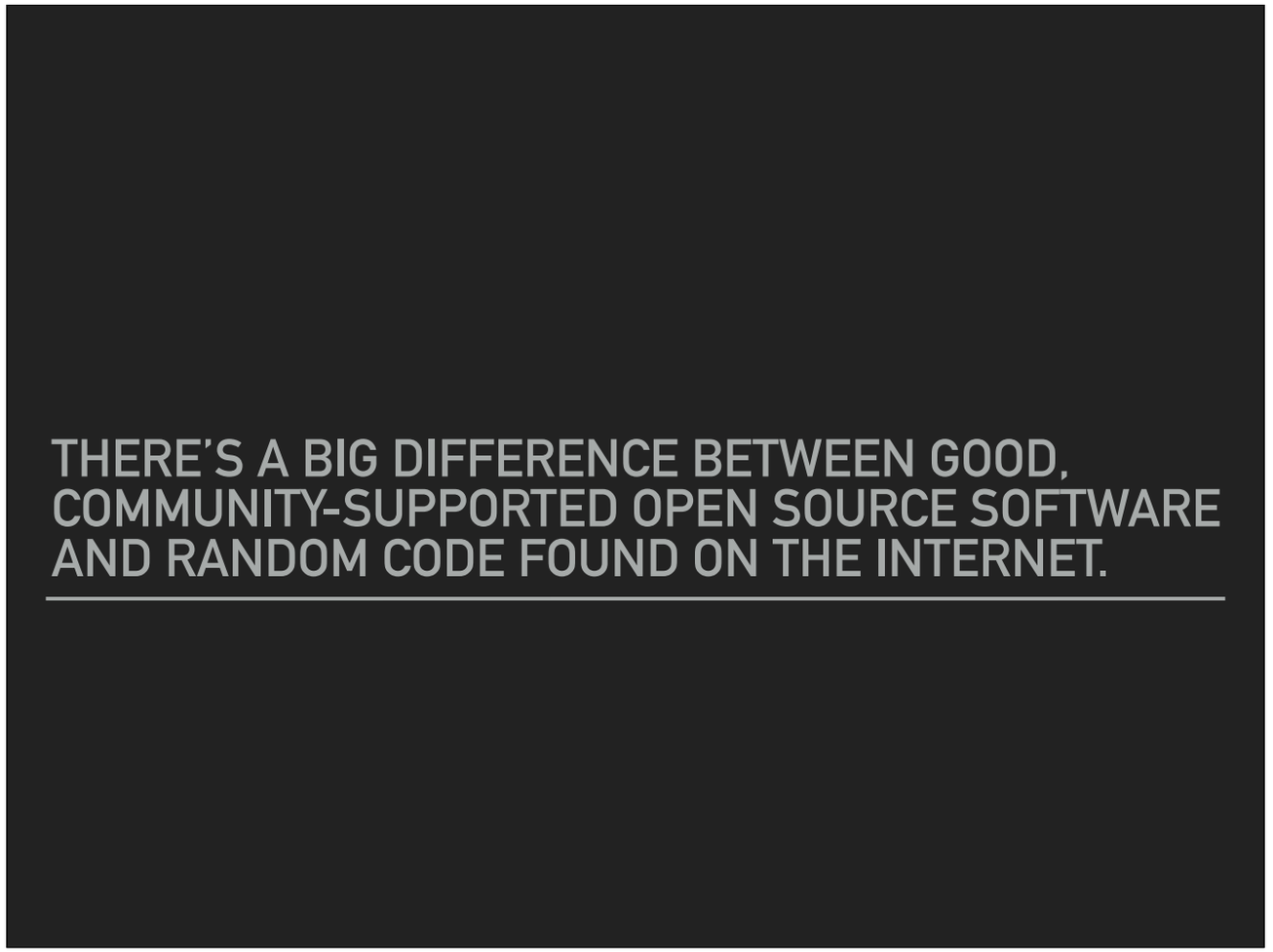
Sometimes as a security professional you are not aware of a product until it is ready for release and the team contacts you to say they have an issue.

Today teams are beginning to be more proactive instead of reactive so hopefully the security team is brought in early during architecture discussions.

I hope this presentation enables you to be more proactive.

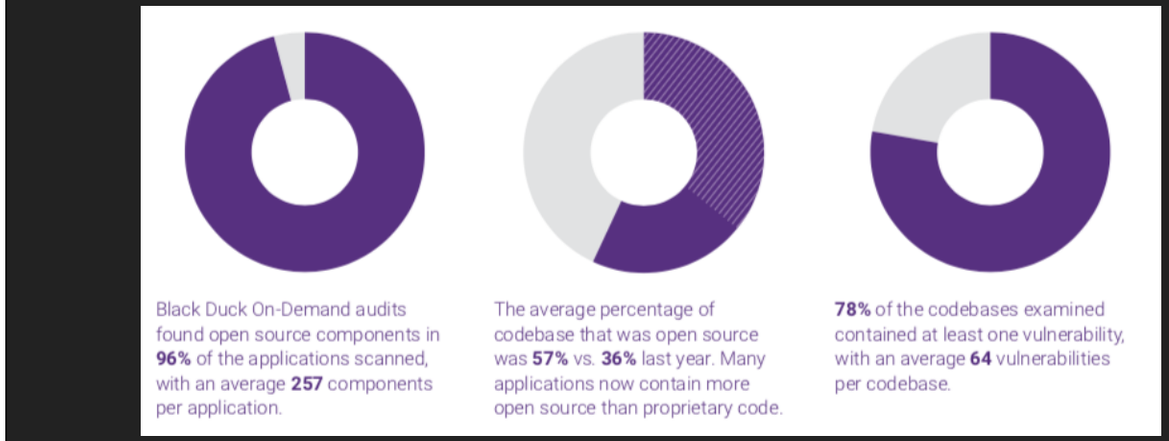
“I AM INVOLVED IN EXAMINING OPEN SOURCE PROJECTS AND THIRD PARTY COMPONENTS FOR 'KNOWN GOOD' DEVELOPMENT PRACTICES.”

I originally was a Security Champion in Intel's Open Source Technology Center (OTC). This was a full time job and when the Security Development Lifecycle was expanded to cover all software at Intel, well let's say it was a nightmare. No one truly understood how much open source software was being used but our small team understood all too well. I moved to the Corporate team responsible for governance, training and SDL. This is important because in order to work across Business Units you need to have management backing.

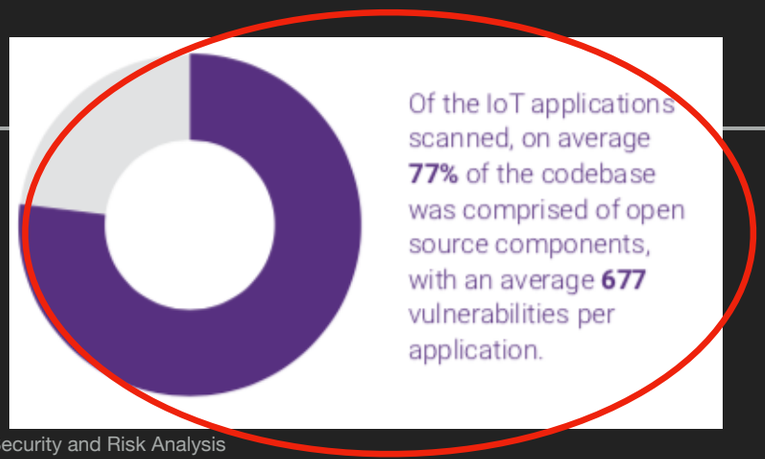


I am leaving these original notes from one of my co-workers in because they are exactly what I was stating in the previous slide notes.

"So, let's be honest, I thought the team at SeCoE that made this policy change was making a mistake. I mean, this was better than doing full SDL on every single bit of code you used, but surely our engineers were good enough at determining what packages were good and which were terrible? Turns out, no, we're not."



YIKES



Source:
2018 Open REPORT Source Security and Risk Analysis
Synopsys Center for Open Source Research & Innovation

This is absolutely not good news at all. In fact it shows that as more open source software is used we are getting worse at securing it.

Source: 2018 Open REPORT Source Security and Risk Analysis
Synopsys Center for Open Source Research & Innovation

I WANT TO ENABLE YOU TO SHIP MORE SECURE BETTER QUALITY OPEN SOURCE SOFTWARE.

By giving you some real world examples and a few tools to use I hope to help instill a security first attitude. This enables you to make good choice up front rather than reacting to issues after releasing the software.

HOW DO I CHOOSE ~~GOOD~~ OPEN SOURCE PACKAGES?

I am not here to tell you how to choose open source. Just here to help you make better choices. So I will explain my process for looking at packages because "GOOD" is such a subjective term.

HOW DO I CHOOSE SECURE OPEN SOURCE PACKAGES?

I will be the first to say you can not ever be sure the open source software you have chosen to incorporate into a project is secure. But by taking some time early in the project you can minimize your risk to security issues during or after your project has been released.

SECTION	GRADE	GRADING GUIDELINES
First look		A - Mentions security audit or other proactive security activity. B - No major warning signs, and code is used professionally. C - No major warning signs, but not widely used or not well-supported. D - Code has minor warning signs that need to be investigated in more detail. F - Code has known issues, major warning signs, or is abandoned
Contributors and activity		A - At least five significant, active contributors. B - More than two significant, active contributors. C - Only one major contributor who is active. D - Project has been inactive for nine months or less. F - Project has been inactive for more than one year.
Security issues		A - Project has had previous security issues and handled them quickly and well. Bonus if they also mention doing proactive security such as fuzz testing, static analysis, or security audits. B - Project has a plan for handling security issues but hasn't had to use it much yet. C - Project does not have a plan for security issues but at least has an active bug tracker and issues get resolved. D - Project does not seem to resolve many open bugs. F - Project has open security issues that are not in the process of being resolved.
Test suite		A - Project has test suite with good coverage of positive and negative test cases set up as part of continuous integration, and test results are published for each build. B - Project has test suite with good coverage but no continuous integration. C - Test suite mostly covers positive test cases; very few or no error cases. D - Test suite has very low coverage or is only a few examples. F - No test suite.

This is so important I moved it to the beginning of the presentation just to make sure we get to talk about it first..

Using this grading card won't guarantee that the team has made a good choice, but it will certainly lower the risk associated with the choices a project team may make.

TAKE A LOOK (ARE THERE RED FLAGS?) REALLY READ IT

CHECK FOR THE NUMBER OF CONTRIBUTORS & ACTIVITY

DOES THE PROJECT HANDLE SECURITY ISSUES

LOOK FOR A TEST SUITE (MAKE SURE IT DOES SOMETHING)

BE AWARE TRUST NOTHING

KEY TAKEAWAYS

So here's the list all on one slide. It might take some time the first few times you run through this list. BUT if you do this every time and take a closer look at everything with a some amount of scrutiny you will lower the projects overall risk exposure.

STEP 1:
TAKE A LOOK

https://clipartxtras.com/categories/view/24dc10f8c0e963e9168b2fcab20d51d21feb8bff/take-a-look-clipart.html

The very first thing we want to do is really take a close look at what we want to use.

- READ THE README,,OR ANY OTHER READILY AVAILABLE INTRODUCTORY INFORMATION?
- DOES THIS CODE APPEAR TO BE HELD TO GOOD SOFTWARE DEVELOPMENT STANDARDS?
- IS THIS CODE USED PROFESSIONALLY OR IS IT A HOBBY PROJECT?
- ARE THERE ANY SIGNS THAT THERE ARE KNOWN ISSUES WITH THIS CODE?
- DOES THIS CODE ONLY SOLVE ONE USE CASE OR IS IT ROBUST ENOUGH FOR OTHER USE CASES?
- IS THIS CODE ACTIVE OR AN ARCHIVE ESSENTIALLY ABANDONED?

LOOK FOR WARNING SIGNS.

SOME KEY QUESTIONS FOR A FIRST LOOK AT A NEW PACKAGE

LET'S LOOK AT SOME WARNING SIGNS

http://chittagongit.com/icon/high-risk-icon-21.html

So let us see how you can add risk to your project when you don't look at what you are adding. These are all in the real world examples and I am not naming and shaming. These developers have given you the information to make an informed decision on whether you want your companies reputation and name brand resting on this code.

"use TweetNaCl.js (a TweetNaCl port to JavaScript) rather than this implementation, which is more likely to perform in constant time and has likely seen more eyes for review/audits."

https://github.com/andi506/crypto-js

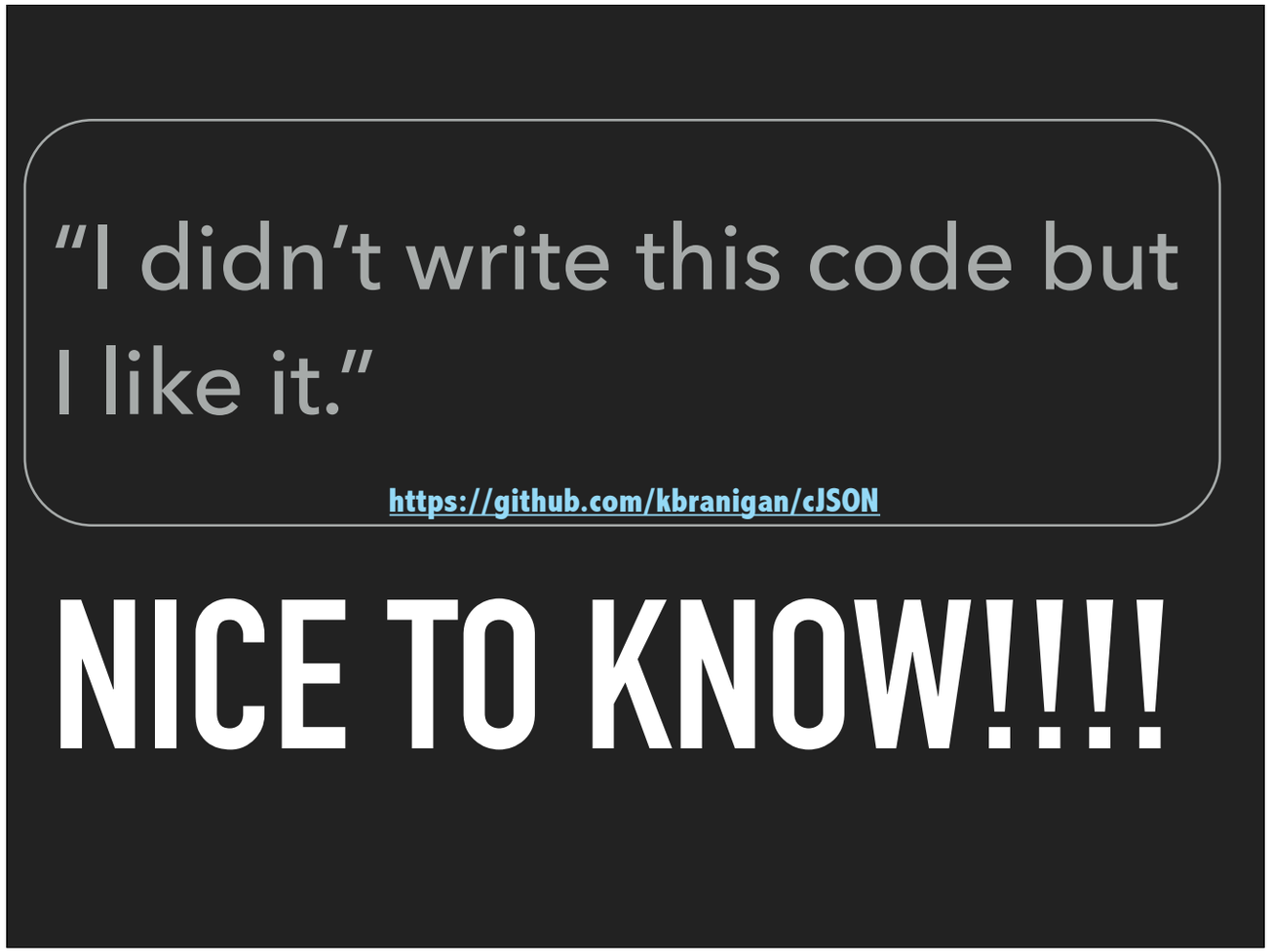
https://github.com/01org/IntelRackScaleArchitecture

**********DISCLAIMER**********

This code is reference software only and is not feature complete. **It should not be used in commercial products at this time.** Intel makes no claims for the quality or completeness of this code.

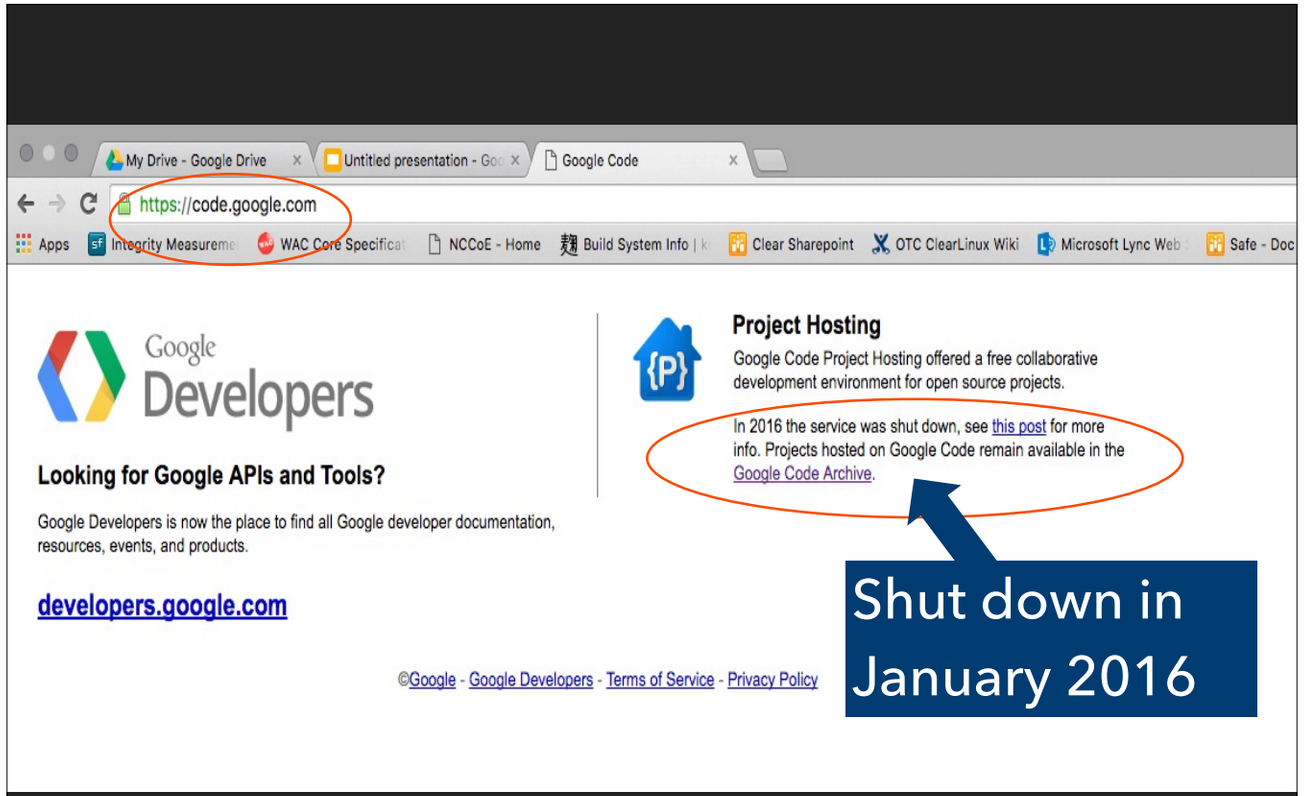
EVEN THE DEVELOPERS SAY TO USE SOMETHING ELSE. . .

Many times developers put out example code or proof of concept code and most (not all) tell you up front. The key here is DON"T put this code into production environments.



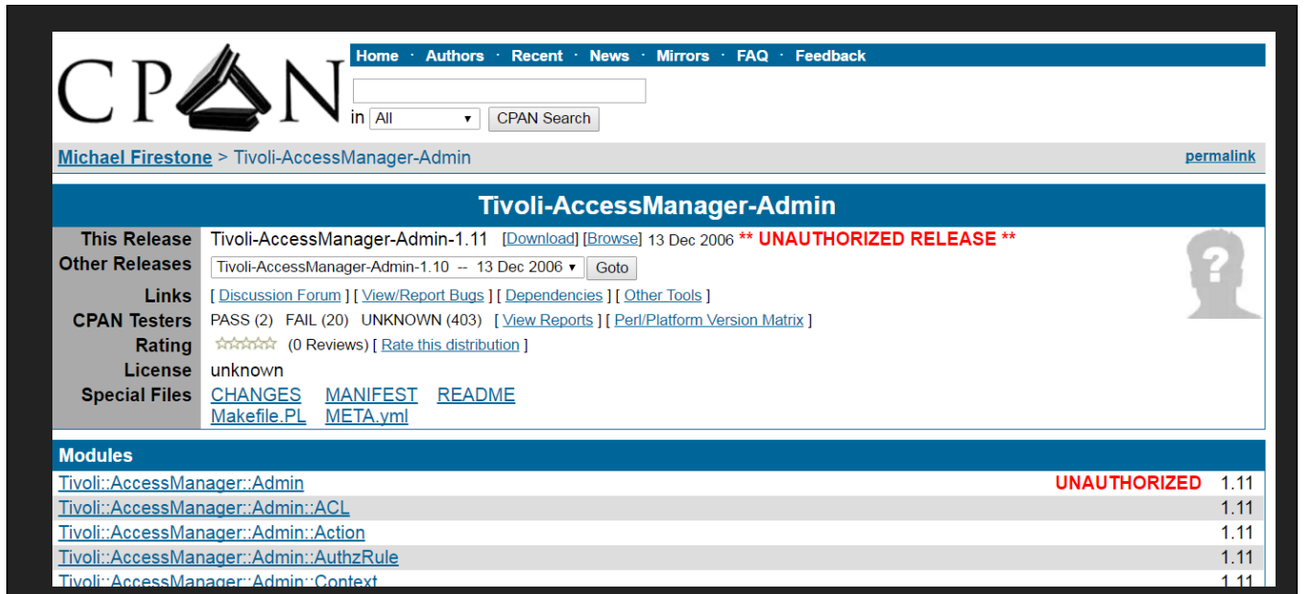
https://github.com/kbranigan/cJSON

This really should be a very big red flag. Where did they get it from? Is the license correct or was it changed? The main take away here if they didn't write it move along and look somewhere else. It is not worth the headaches.



In 2016 code.google.com was shut down. Many projects were moved elsewhere and are still maintained find those repositories, don't use anything here.

Archived code is dead code... Don't use it. No one is looking at it or fixing issues..



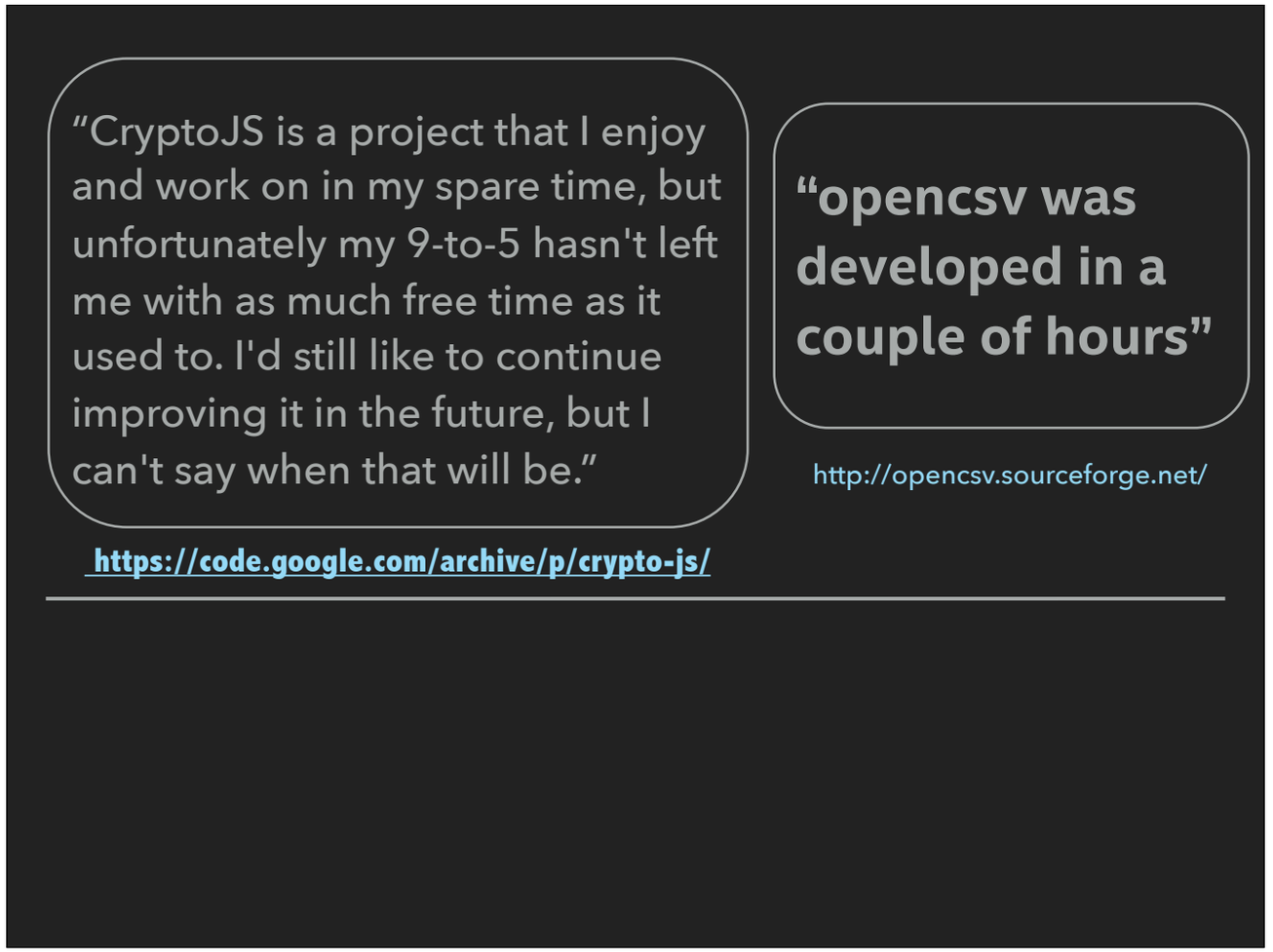
DOES ANYONE REALLY LOOK TO SEE WHAT THIS MEANS?

ThisFrom CPAN:

- A co-maintainer uploaded a new release, but because of an oversight wasn't granted permission on one of the modules. This often happens with distributions that have a different release manager each cycle.
- **Someone without co-maintainer permissions forked the distribution and uploaded it.**
- An author makes a new release with a new namespace without realizing that namespace is taken by another author.

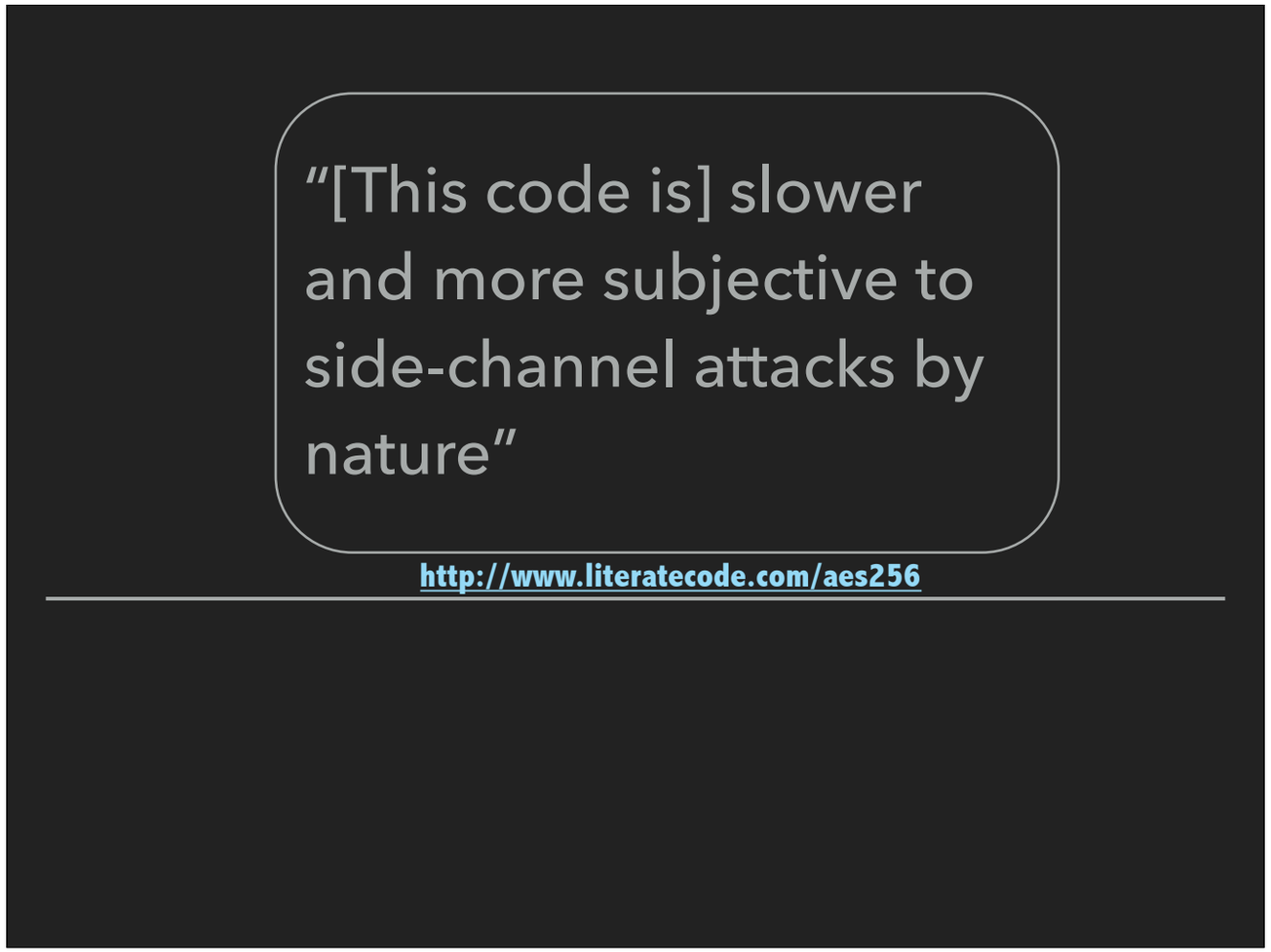
https://www.gograph.com/vector-clip-art/nope.html

Provenance is everything. Where did it come from and can you trust it.



Two examples of developers that are telling you they don't necessarily do this for a living and maintenance will be less than optimal if at all. If a vulnerability is discovered can you trust that it would get fixed in a timely manner.. Probably better to look elsewhere. NOTE: The statement on opencsv would not be found without reading to the very end of the description to "Who maintains opencsv?"

Don't hide away from bringing these issues forward, finding these things out after the fact is far worse than making it known this has huge implications.



Someone searching for a fast implementation of aes wanted to use this. The author tells you it was built to solve one use case and also goes on to tell you it is subject to attack vectors. I am not sure I would want to rest my reputation on adding this to a project.

“cJSON aims to be the dumbest possible parser that you can get your job done with. ”

https://github.com/kbranigan/cJSON/commit/730209a718cc9bada631cea136d13017752720f5

When it comes to parsers the minute someone suggests writing their own just say NOPE. When you see key words like faster, lightweight, small footprint realize that something had to be given up to gain those attributes.

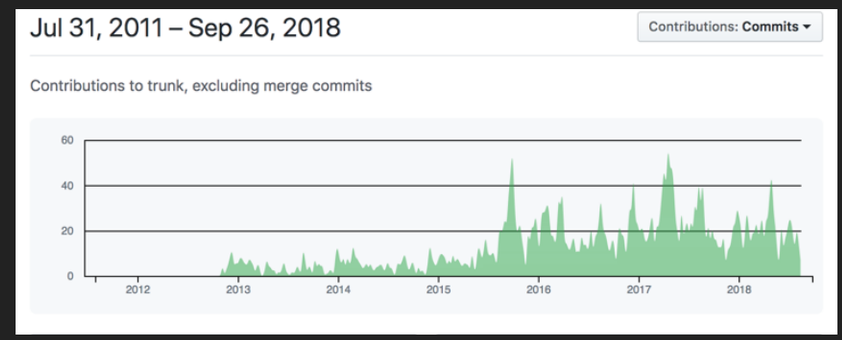
STEP 2:
CHECK THE CONTRIBUTORS AND COMMUNITY

Github has some great metrics that can really help you out.

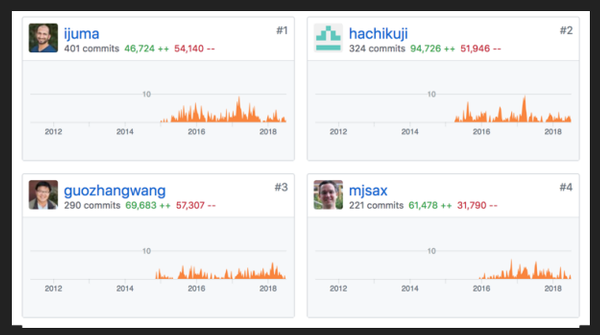
- HOW MANY ACTIVE AND SIGNIFICANT CONTRIBUTORS?
- IS THIS CODE ACTIVELY MAINTAINED OR IS IT ABANDONED?
- HOW MANY PULL REQUESTS & CHECKINS IN THE PAST YEAR?
- ARE ISSUES FIXED AND RELEASED ON A REGULAR BASIS?
- WHO SIGNS OFF ON CODE REVIEWS?
- IS THERE MORE THAN ONE MAINTAINER?

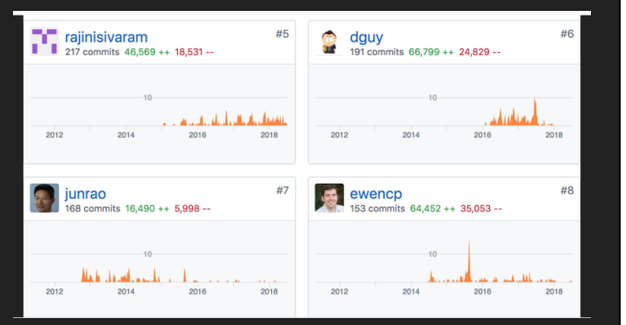
KEY QUESTIONS ABOUT CONTRIBUTORS

Active contributors are good but it the significant contributors that makes a big difference. When was the last time anything was checked in or when was the last release? How many outstanding pull requests? Are there discussions taking place? How many accepted pull requests?Does the project fix issues and issue new releases? It does no one any good if there is a fix checked in but no release. Do code reviews happen? Is it one person or a group effort? More eyes is better. Than of course is there more than one maintainer. More maintainers the better.



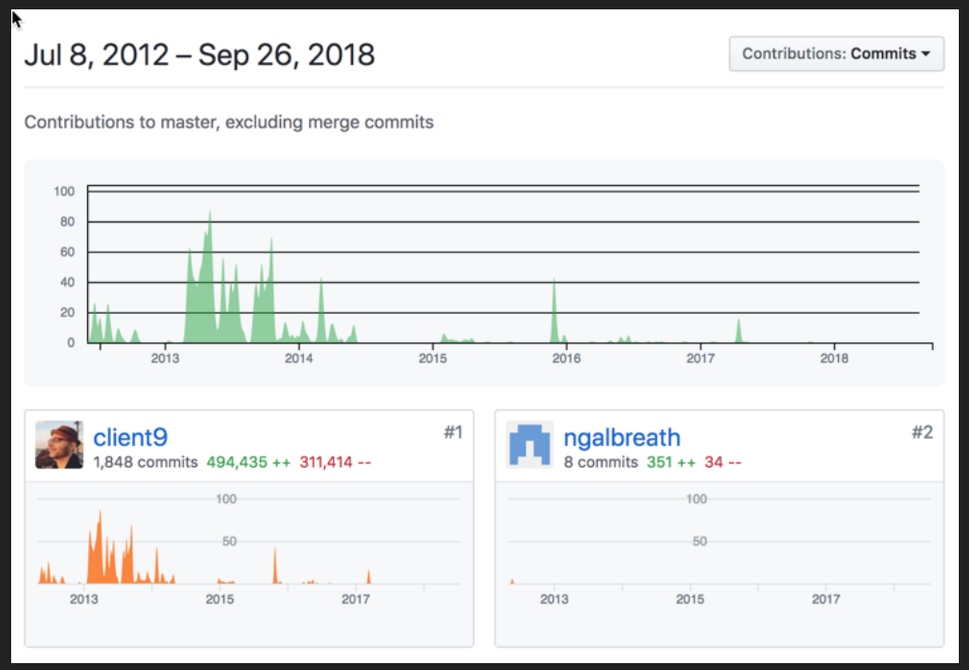
Good example: many active contributors





So here’s an example of an active project. Apache Kafka. Many Active and significant contributors

A good example of one you might want to skip: one significant contributor and not recently active



There were 18 contributors, 14 submitted significantly < 50 lines. Almost all of the code was written by one contributor and has had no activity since early 2017. Since it is a parser I would probably look somewhere else.

STEP 3:
CHECK HOW THEY HANDLE VULNERABILITIES

- IS THERE A CLEAR WAY TO REPORT SECURITY VULNERABILITIES?
- IS THERE EVIDENCE THAT VULNERABILITIES ARE FIXED IN A TIMELY MANNER?
- IS THERE ANY EXPLANATION OF WHAT HAPPENS WHEN A SECURITY ISSUE IS REPORTED?

KEY QUESTIONS ABOUT HANDLING SECURITY ISSUES

An ideal procedure should involve a way to keep the vulnerability secret until a fix is found. Typical good solutions: send email to a special security mailing list. Bug tracker with special “security” flag. If there’s no way to report security issues specifically, assume they have not thought about it (and this is a bad sign).

HTTPS://WWW.APACHE.ORG/SECURITY/

GREAT EXAMPLE:

So one of the best examples is Apache. They have a very clear vulnerability process that we know from experience that they follow, and it applies to all the active projects under the Apache banner.

VULNERABILITY HANDLING

A typical process for handling a new security vulnerability is as follows. Projects that wish to use other processes MAY do so, but MUST clearly and publicly document their process and have security@ review it ahead of time.

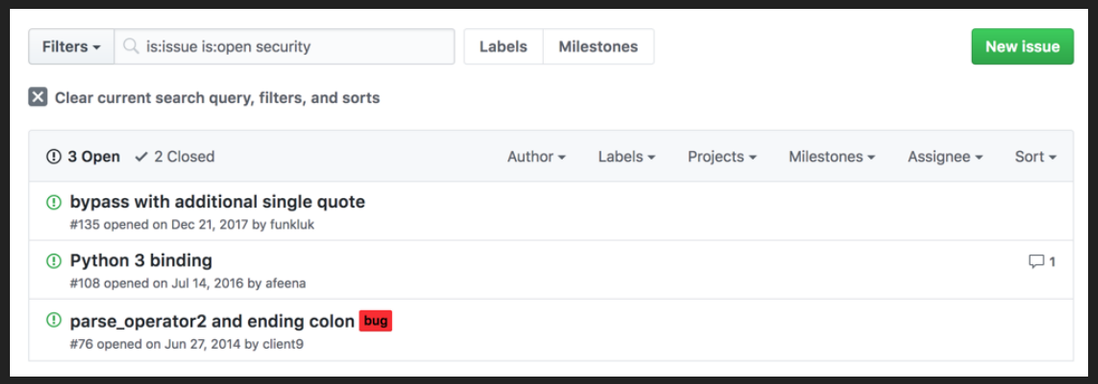
Note: No information should be made public about the vulnerability until it is formally announced at the end of this process. That means, for example that a Jira issue must NOT be created to track the issue since that will make the issue public. Also the messages associated with any commits should not make ANY reference to the security nature of the commit.

1. The person discovering the issue, the reporter, reports the vulnerability privately to security@project.apache.org or to security@apache.org
2. Messages that do not relate to the reporting or managing of an undisclosed security vulnerability in Apache software are ignored and no further action is required.
3. If reported to security@apache.org, the security team will forward the report (without acknowledging it) to the project's security list or, of the project does not have a security list, to the project's private (PMC) mailing list.
4. The project team sends an e-mail to the original reporter to acknowledge the report.
5. The project team investigates report and either rejects it or accepts it.
6. If the report is rejected, the project team writes to the reporter to explain why.
7. If the report is accepted, the project team writes to report to let them know it is accepted and that they are working on a fix.
8. The project team requests a CVE number from security@apache.org by sending an e-mail with the subject "CVE request for _" and providing a short (one line)

HTTPS://WWW.APACHE.ORG/SECURITY/COMMITTERS.HTML

WRITTEN POLICY

They even go one step further to let people know what to expect when they report a bug, so you can tell if they're handling things by their process pretty easily.



LOOK FOR UNFIXED SECURITY BUGS

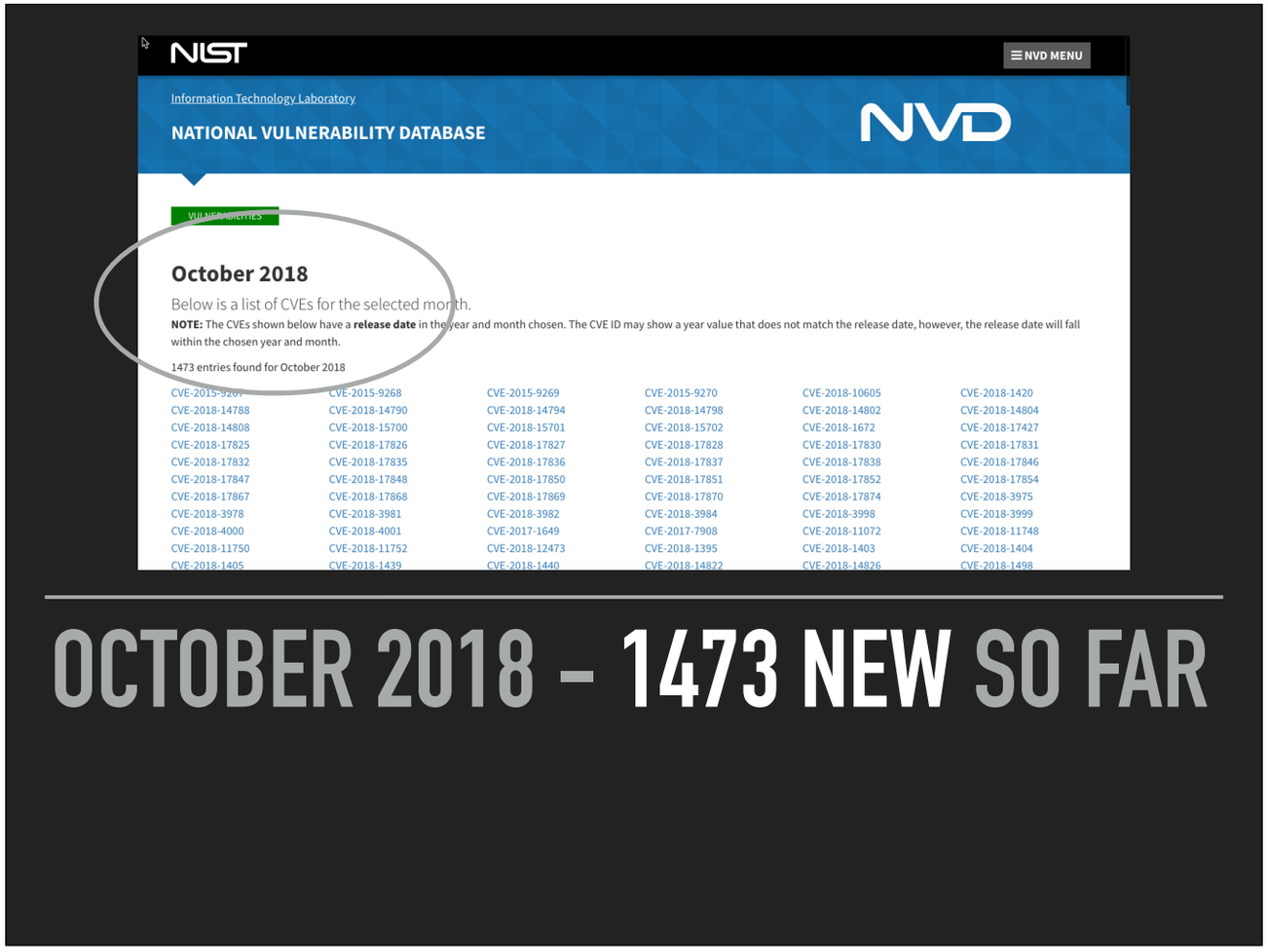
Unfixed security bugs will come back to haunt you someday sometime. Better to know up front and choose something totally different.

No known vulnerabilities does not equal security

Just because there are no known vulnerabilities does not mean that it is safe to use. In fact it means quite the opposite. You should probably scrutinize it even more because it more than likely means no one is really looking at it. It does not matter how well looked at it will have vulnerabilities and you need to be prepared to remediate them.

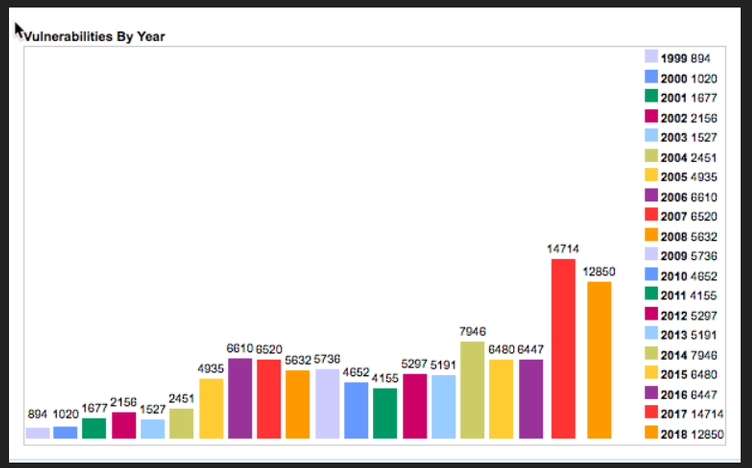
- CVES (COMMON VULNERABILITIES AND EXPOSURES) AREN'T EASY TO FILE,
- THIS CAN BE A SIGN OF A LACK OF SECURITY EXPERTISE
- MANY TIMES NO ONE IS LOOKING
- SOMETIMES ISSUES ARE ACTIVELY REJECTED OR HIDDEN
- HOW PROJECT TEAMS REACT TO KNOWN VULNERABILITIES WILL HELP YOU TO EVALUATE THEIR SECURITY PROCESSES.

WHY?



OCTOBER 2018 – 1473 NEW SO FAR

At the time of slide creation 1032 vulnerabilities just for the month of September 2018.



https://www.cvedetails.com/browse-by-date.php

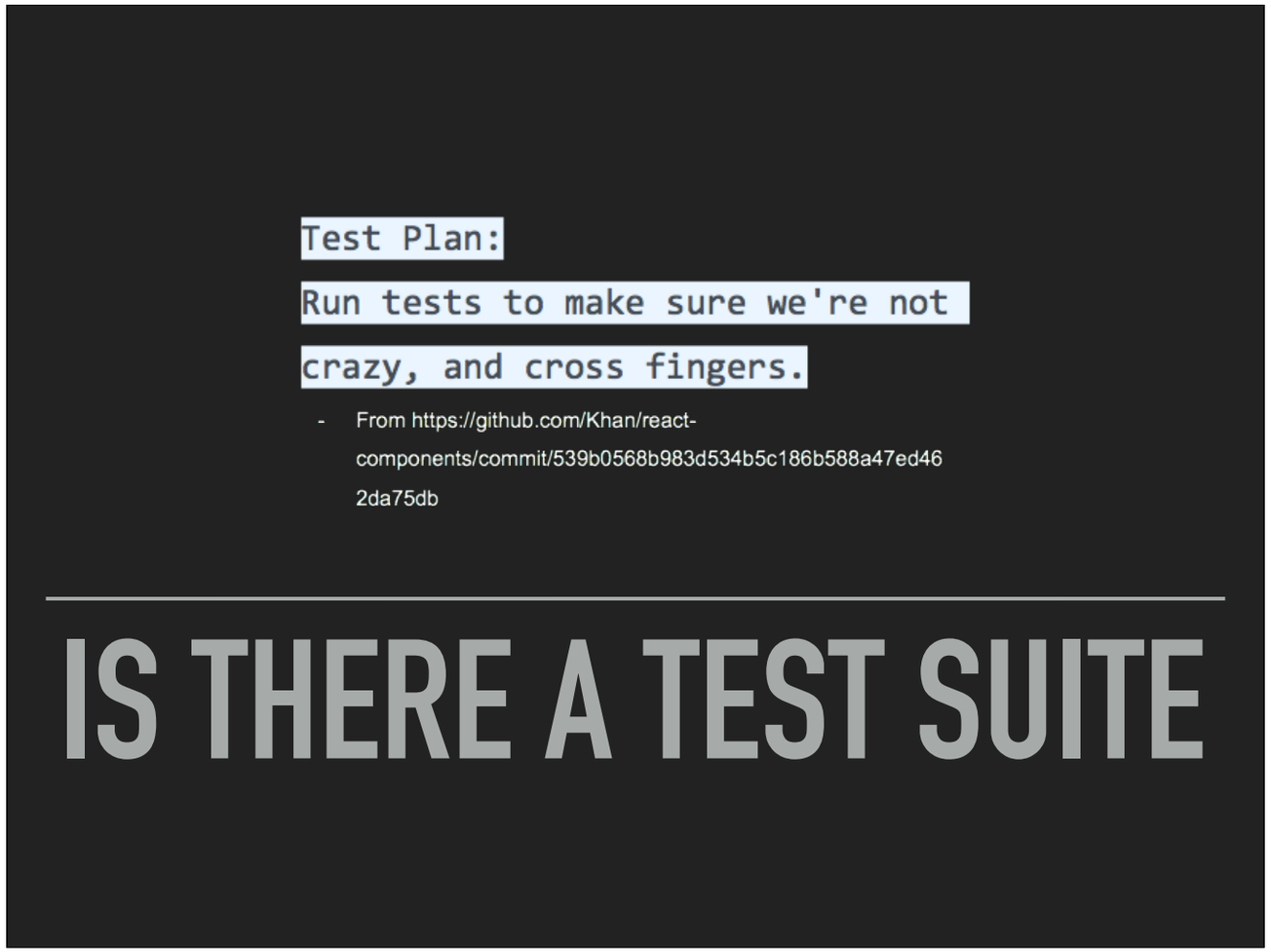
IT SEEMS THAT WE ARE GETTING WORSE AT SECURITY

Actually it is that there are more looking for vulnerabilities and more software usage in the world. Last year the total number of vulnerabilities was more than the first 6 yrs of tracking combined.

https://www.cvedetails.com/browse-by-date.php

STEP 4:
IS THERE ANY TESTING

Another thing you can do if you're just not sure about a project is to take a look at the test suite.



So what do good and bad test suites look like? I was going to show some examples, but it got unreadable pretty quickly, so let's just talk qualities. [Speaker will summarize from slide]

TESTING ISN'T THE ONLY THING WE TAKE INTO ACCOUNT, BUT IT CAN BE USED AS A RULE OF THUMB IF YOU DON'T KNOW SECURITY AND WANT TO GUESS AT WHAT MIGHT BE A GOOD LIBRARY.

- DOES THIS TEST SUITE COVER BAD BEHAVIOUR?
- HOW COMPREHENSIVE IS THIS TEST SUITE?
- DO ALL TESTS PASS?
- IS THERE CONTINUOUS INTEGRATION FOR TESTS?

TESTING IS ESPECIALLY IMPORTANT FOR LIBRARIES THAT HANDLE USER INPUT: PARSERS, INPUT VALIDATION LIBRARIES, ETC. A POOR TEST SUITE DOESN'T GUARANTEE A BLACKLISTED COMPONENT, BUT A GOOD SUITE OFTEN IMPLIES A BETTER CHOICE.

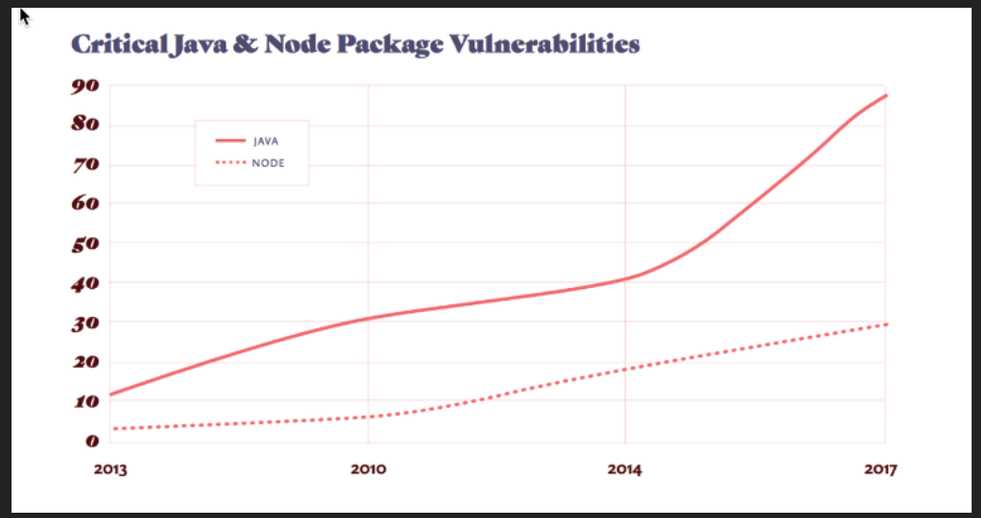
KEY QUESTIONS ABOUT TEST SUITES

As before, here's some key questions for folk who aren't sure what a "good" test suite looks like. We're looking for a good variety of tests covering good and bad behavior, and test suites are really important for things that parse user input.

STEP 5:
BE AWARE

And finally, I want to take a moment to make sure you're all aware of the type of assumptions we see that result in us making poor choices.

Good packages do not guarantee good dependencies



“90% of tested organizations use vulnerable dependencies."

- https://snyk.io/

- UPSTREAM PROJECTS DO NOT USE THE SAME CRITERIA FOR INCLUSION AS PRODUCTS SHOULD
- IT IS POSSIBLE FOR A PACKAGE TO DEPEND ON CODE THAT HAS VULNERABILITIES

UNDERSTAND YOUR DEPENDENCIES AND THE ADDITIONAL RISK IT BRINGS. ALLOCATE TIME TO WORK ON THE ISSUES

WHY?

Why is this an issue? Remember that when you carefully vette an OSS component that you really need to be aware of what it depends on. Because when you bring that component into your project you will also be bringing anything it depends on too, good or bad.

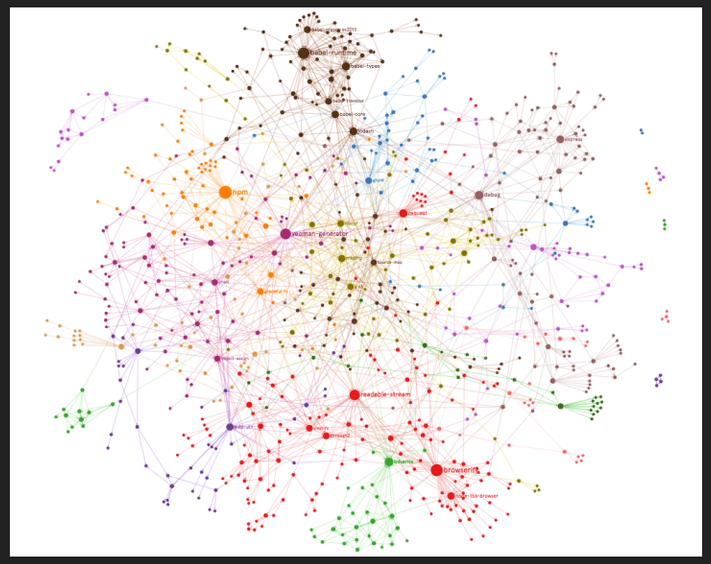
NPM (NODE.JS), PYPI (PYTHON), RUBY GEMS (RUBY), CPAN (PERL), ETC. ARE NOT CURATED

ALL REPOSITORIES ARE NOT CREATED EQUAL

Anybody can create and check in to projects. Just because you have valid credentials to check in to a project does not mean you're a good guy.

Python Security - https://github.com/PyCQA/bandit
npm Security - https://github.com/i0natan/nodebestpractices#6-security-best-practices

I wonder what happens when you remove a module?



Oh wait!! that already happened

Package managers don’t always imply high quality

As seen in this chart of the npm tree of the first 4 levels of dependency shows that if you were to slip one module in there or even take one away you could wreak havoc

Oh wait!! that already happened

A programmer named Azer Koçulu, during a trademark dispute, refused to change the name of his software module. Lawyers were able to get npm to give them rights to the module. In protest Koçulu removed his other code from npm, including something called npm left-pad which was used by thousands of projects.

“Popular software projects like Babel, which helps Facebook, Netflix, and Spotify run code faster and React, which helps developers build better interfaces, were suddenly broken and no more work could be done with them.”

Npm re-published the code, giving it to a new owner

“HOW REMOVING 11 LINES OF CODE NEARLY BROKE THE INTERNET”

A related story about this is the “11 lines of code that nearly broke the Internet”
This is kind of a sad story about a guy whose open source project had the same name as something someone else had trademarked. When he refused to change the name of his already well-established project, the lawers got npm to give them rights to the name and thus trounced his work.
Upset, he pulled the rest of his code in protest, including software that was used by a bunch of big names, including Facebook and Netflix. All of a sudden, big popular projects were broken, and people were freaking out. Npm re-published his removed code and gave it to a new owner.

There’s two things you should worry about here: one, if there’s a trademark dispute on a name, code you’re using may suddenly be completely different code. And similarly, if a project becomes popular, it could be handed over to a new owner without consent or any warning. If you don’t verify the code you’re getting, you could seriously be getting anything, at any time, and not just if the developer decided to become malicious.

BRINGING IT ALL TOGETHER

Lot's of information presented very quickly. I will make this presentation available to all including the presenter notes. Let's take a look at what you should take away at the least.

Take a look (Are there red Flags?) Really Read it
Check for the number of contributors & activity
Does the project handle security issues
Look for a test suite (make sure it does something)
Be aware Trust nothing

Assumptions and poorly vetted software choices are common. By taking a little bit of time up front you can minimize the risk of using poorly managed open source.

Thank you

LinkedIn: **https://www.linkedin.com/in/mikide/**

Twitter: **@theDawgCr8**

Email: sec-princess@unroutable.me

Intel Credits: Terri Oda, Tiberius Heflin, John Anderson

https://github.com/sec-princess/Seagl-20181109

LinkedIn: https://www.linkedin.com/in/mikide/
Twitter: @theDawgCr8
Email: sec-princess@unroutable.me

Resources:

https://github.com/sec-princess/Seagl-20181109
https://snyk.io
https://nvd.nist.gov/vuln/full-listing
https://www.cvedetails.com/