

Security Jam

Choosing Better Open Source

presented by Miki (Michaela) Demeter
Security Researcher at Intel Corporation

<http://go/securityjam>

Thursday, April 18th

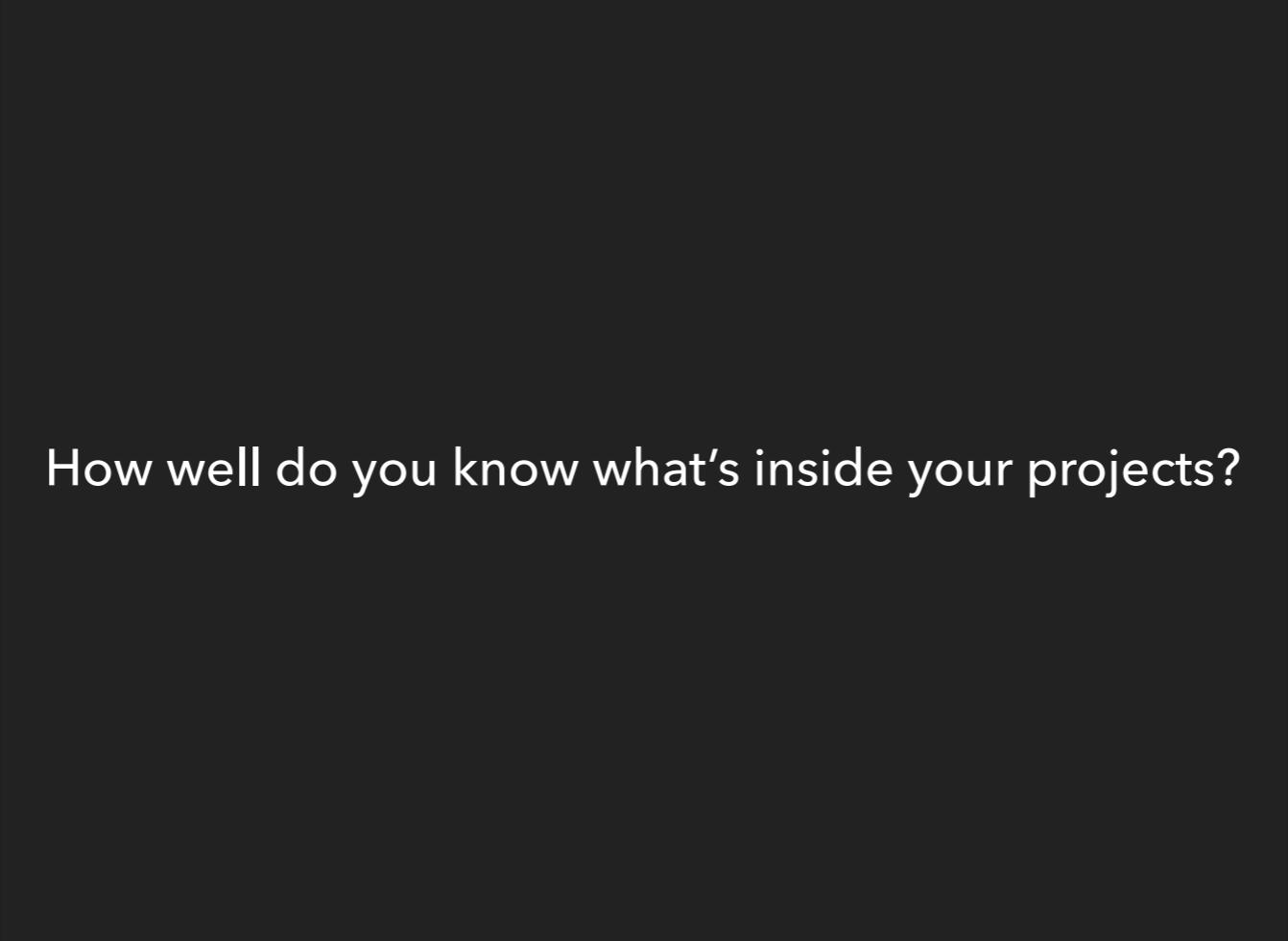
- NA-**EMEA** Session
9am Thursday Pacific; 17:00 Thursday London
- NA-**APAC** Session
6pm Thursday Pacific; 9:00 Friday Singapore





F5 SECURITY JAM

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



How well do you know what's inside your projects?

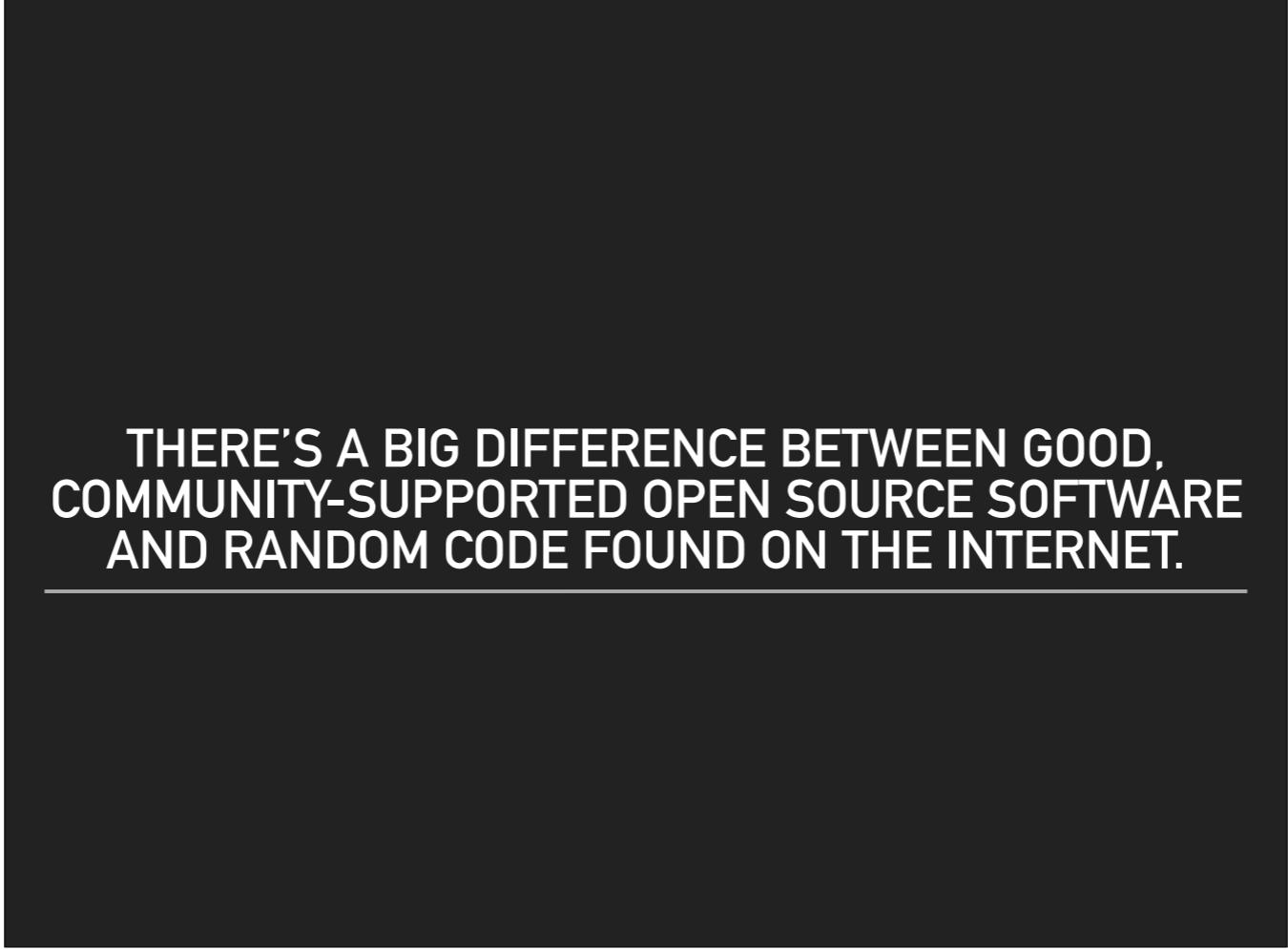
only by examining the components in detail will you know what is inside the project

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 you are brought in early during architecture discussions.

"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 to 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.



**THERE'S A BIG DIFFERENCE BETWEEN GOOD,
COMMUNITY-SUPPORTED OPEN SOURCE SOFTWARE
AND RANDOM CODE FOUND ON THE INTERNET.**

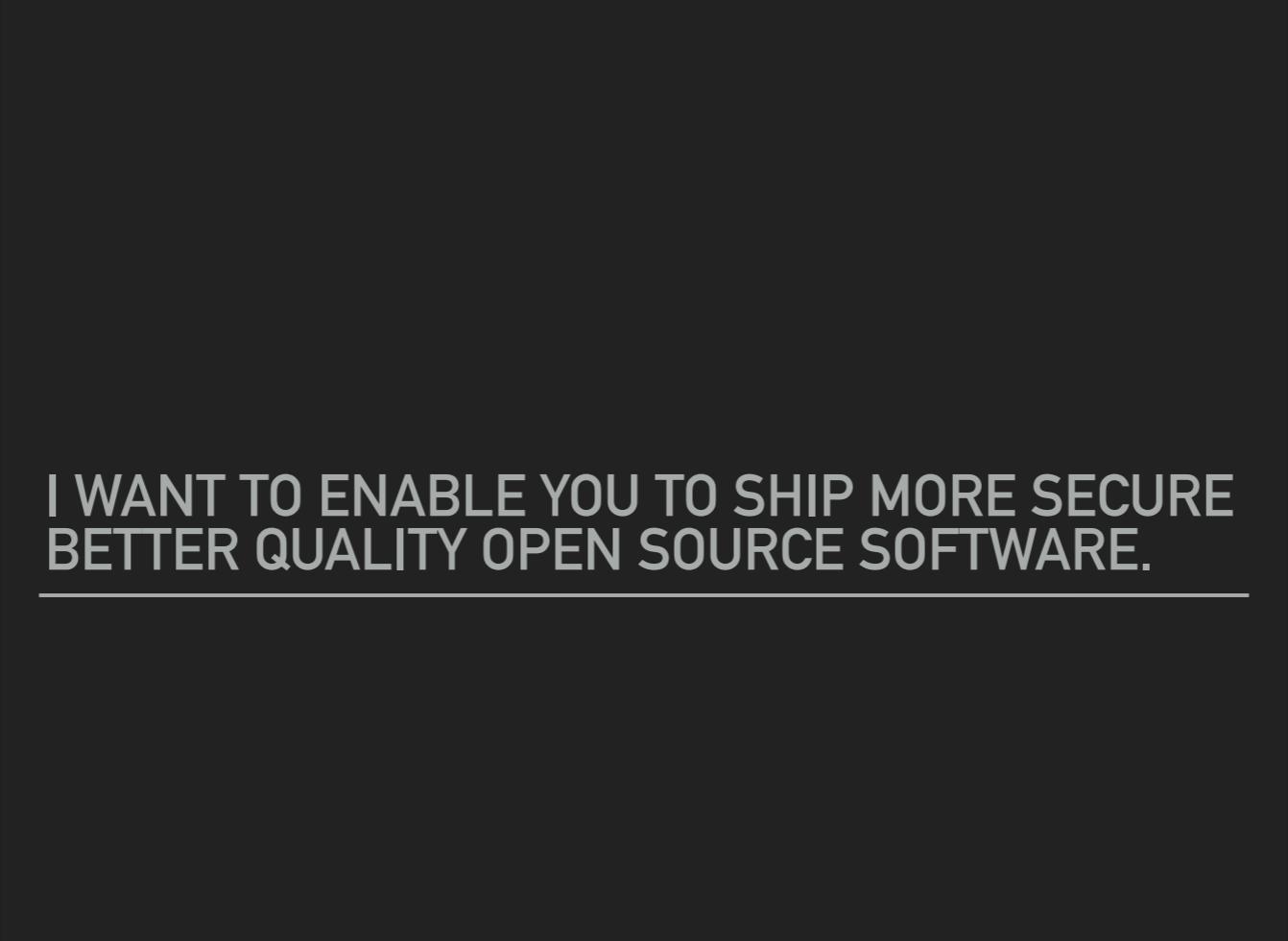
I am leaving these original notes from one of my co-workers in because they are exactly what I was stating in the previous slide notes.

"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."



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.

BEFORE YOU START LOOKING AT CODE OR DEV PRACTICES

- ▶ Choose your repository
- ▶ Look at the license
 - ▶ Is it compatible
 - ▶ Is there a license at all

If this fails you won't need worry about the rest

I can not emphasize this enough. See this all the time where a developer has chosen a repository that was forked and contains code of dubious origin.

License is always a big one when you find out that you have shipped an incompatible license with your product



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

IS IT



IN OTHER WORDS . . .

Basically this is what you are trying to determine in its most basic form.



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>

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.

*****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.

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

READ CAREFULLY & 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.

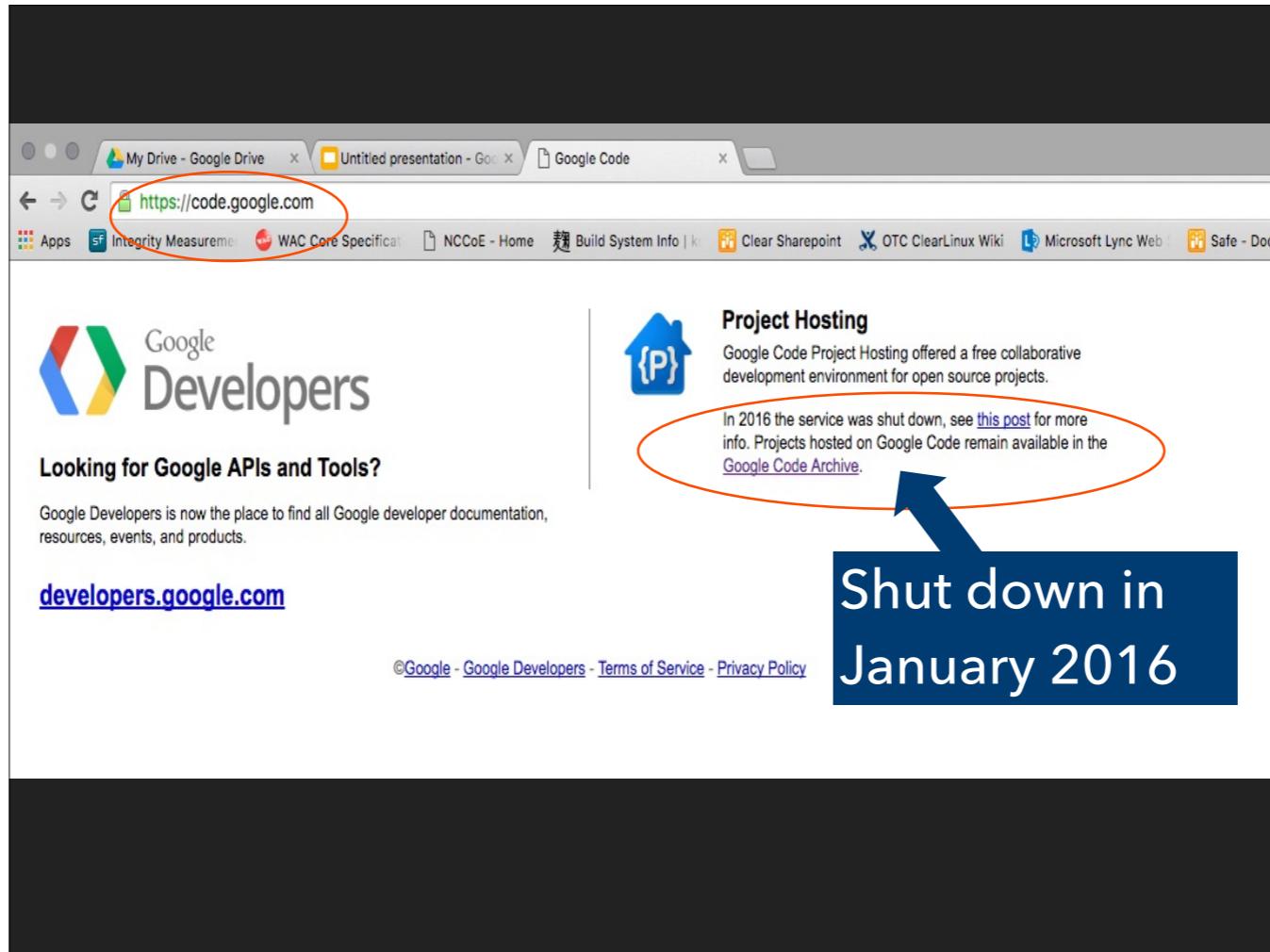
“I didn’t write this code but
I like it.”

<https://github.com/kbranigan/cJSON>

NICE TO KNOW!!!!

<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.

CPAN

Home · Authors · Recent · News · Mirrors · FAQ · Feedback

in All CPAN Search

Michael Firestone > Tivoli-AccessManager-Admin

permalink

Tivoli-AccessManager-Admin

This Release Tivoli-AccessManager-Admin-1.11 [Download] [Browse] 13 Dec 2006 ** UNAUTHORIZED RELEASE **

Other Releases Tivoli-AccessManager-Admin-1.10 -- 13 Dec 2006 Goto

Links [Discussion Forum] [View/Report Bugs] [Dependencies] [Other Tools]

CPAN Testers PASS (2) FAIL (20) UNKNOWN (403) [View Reports] [Perl/Platform Version Matrix]

Rating ★★★★☆ (0 Reviews) [Rate this distribution]

License unknown

Special Files CHANGES MANIFEST README
Makefile.PL META.yml

Modules

Tivoli::AccessManager::Admin	UNAUTHORIZED	1.11
Tivoli::AccessManager::Admin::ACL		1.11
Tivoli::AccessManager::Admin::Action		1.11
Tivoli::AccessManager::Admin::AuthzRule		1.11
Tivoli::AccessManager::Admin::Context		1.11

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.

"CryptoJS is a project that I enjoy and work on in my spare time, but unfortunately my 9-to-5 hasn't left me with as much free time as it used to. I'd still like to continue improving it in the future, but I can't say when that will be."

<https://code.google.com/archive/p/crypto-js/>

"opencsv was developed in a couple of hours"

<http://opencsv.sourceforge.net/>

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 finding these things out after the fact. Make it known this has huge implications.

"[This code is] slower
and more subjective to
side-channel attacks by
nature"

<http://www.literatecode.com/aes256>

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.

WHAT WOULD WE LIKE TO SEE?

We have now looked at some timeless real life examples of everything I hope to never see again. So on to look at some things that help us determine how well the community supports the open source you are choosing to put into a product.

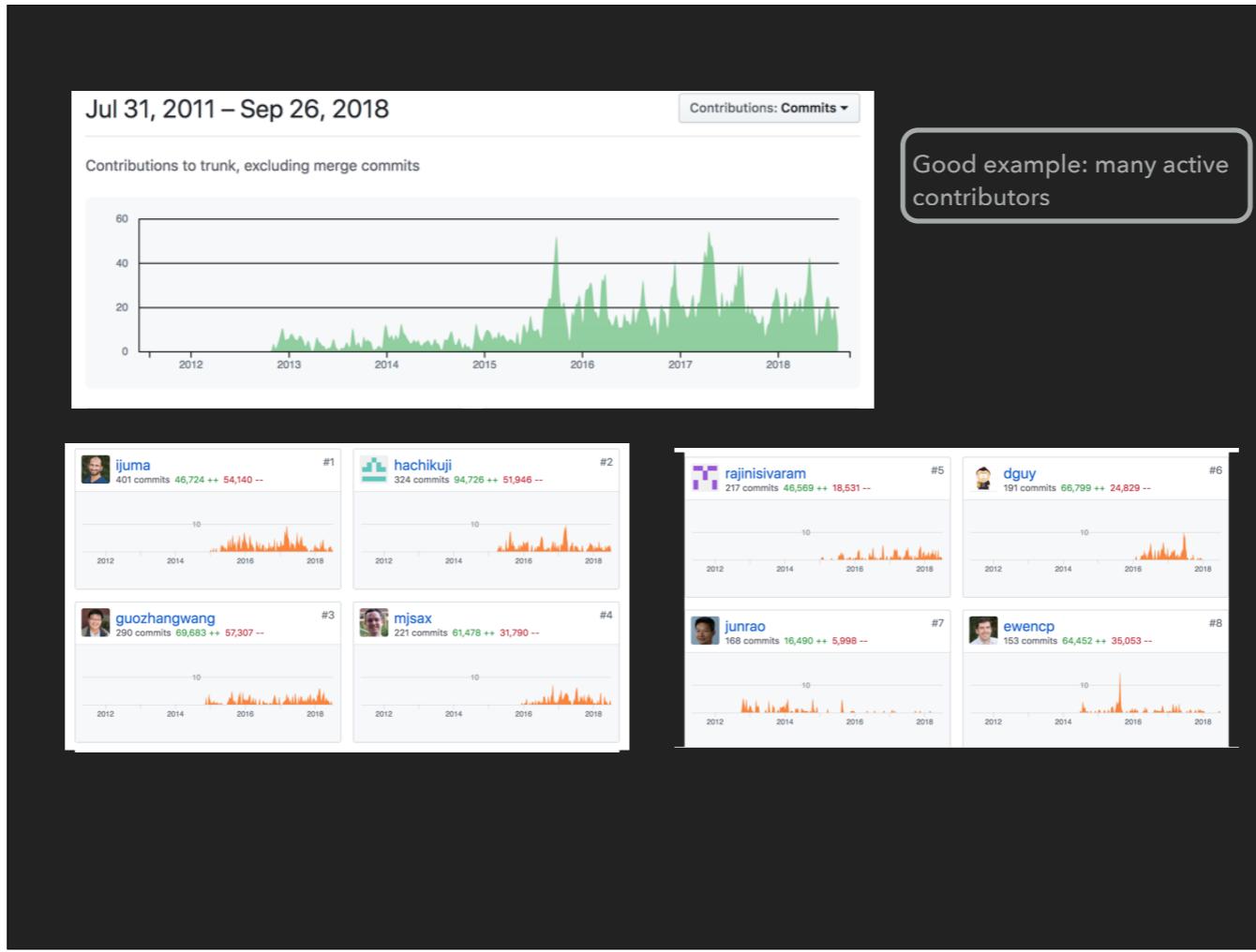
STEP 2: CHECK THE CONTRIBUTORS AND COMMUNITY

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?
-

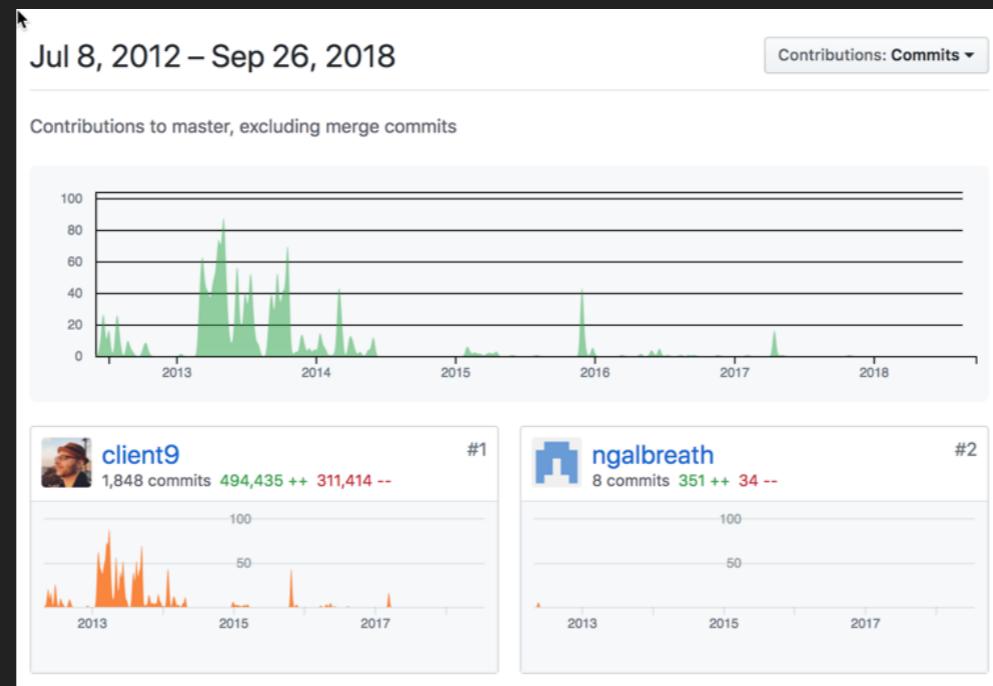
KEY QUESTIONS ABOUT CONTRIBUTORS

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.



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

A good example: 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:
DEPENDENCIES **DEPENDENCIES** **DEPENDENCIES**

Good packages do not guarantee good dependencies

Another thing that comes up a lot is that you've chosen a great, well-supported framework so you think for sure everything they've chosen is also going to be great and well-supported. I also wish this were true, but it's not.

- 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 vet 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

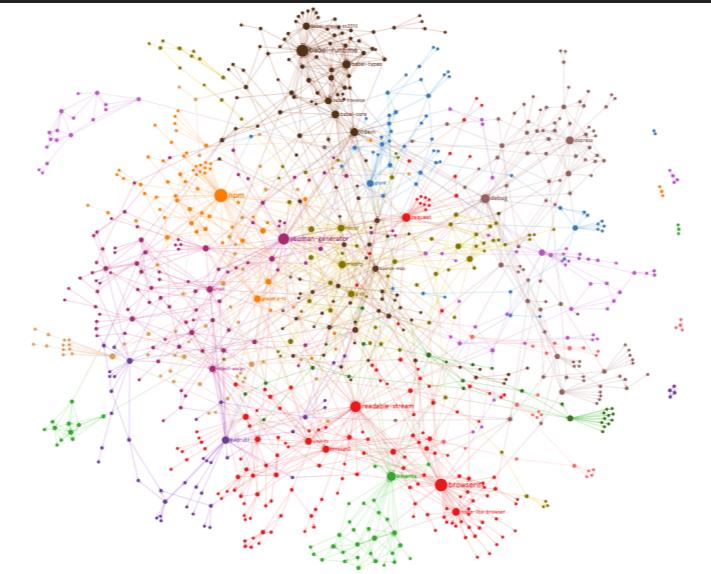
ALL REPOSITORIES ARE NOT CREATED EQUAL

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

We all heard about this one

“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 lawyers 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.

This attack started out as a social engineering attack. The attacker, posing as a maintainer, took over maintainership of the event-stream module.

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

EVENT-STREAM INCIDENT

On the morning of November 26th, npm's security team was notified of a malicious package that had made its way into event-stream, a popular npm package. After triaging the malware, npm Security responded by removing flatmap-stream and event-stream@3.3.6 from the Registry and taking ownership of the event-stream package to prevent further abuse.

Now that it has happened once do you think it can happen again

STEP 4:
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).



THE APACHE SECURITY TEAM

The Apache Security Team exists to provide help and advice to Apache projects on security issues and to provide co-ordination of the handling of security vulnerabilities. All members of the Security Team are also [members](#) of the Apache Software Foundation.

REPORTING A VULNERABILITY

We strongly encourage folks to report security vulnerabilities to one of our private security mailing lists first, before disclosing them in a public forum.

A [list of security contacts for Apache projects](#) is available. If you can't find a project specific security e-mail address and you have an undisclosed security vulnerability to report then please use the general security address below.

Please note that the security mailing lists should only be used for reporting undisclosed security vulnerabilities in Apache products and managing the process of fixing such vulnerabilities. We cannot accept regular bug reports or other security related queries at these addresses. All mail sent to these addresses that does not relate to an undisclosed security problem in an Apache product will be ignored.

Also note that the security team handles vulnerabilities in Apache products, not running ASF services. All reports of vulnerabilities in running ASF services should be sent to root@apache.org only.

The general security mailing list address is: security@apache.org. This is a private mailing list and only members of the Apache Security Team are subscribed.

[HTTPS://WWW.APACHE.ORG/SECURITY/](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.

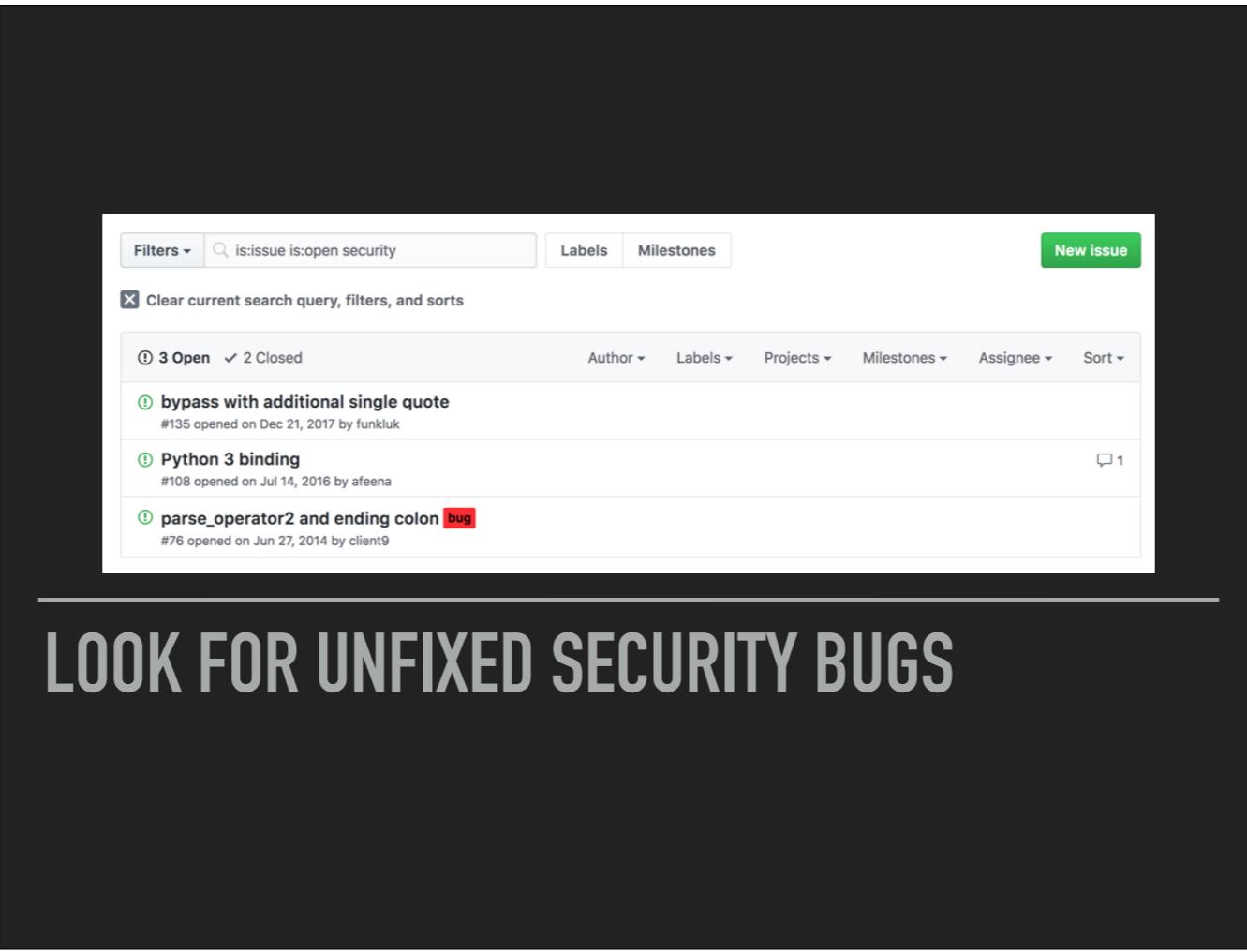
1. The person discovering the issue, the reporter, reports the vulnerability privately to security@project.apache.org or to security@apache.org
2. Messages that do not relate to the reporting or managing of an undisclosed security vulnerability in Apache software are ignored and no further action is 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, if 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 reporter 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](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.

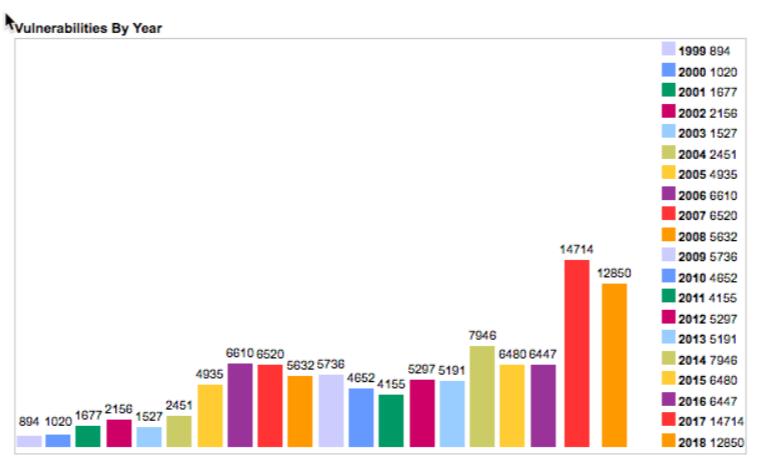
The screenshot shows a web browser displaying the NIST National Vulnerability Database (NVD) at the URL <https://nvd.nist.gov/vuln/full-listing/2019/3>. The page header includes the NIST logo and the text "Information Technology Laboratory" and "NATIONAL VULNERABILITY DATABASE". A green button labeled "VULNERABILITIES" is visible. A large gray oval highlights the section titled "March 2019". Below it, a note states: "Below is a list of CVEs for the selected month. NOTE: The CVEs shown below have a release date in the year and month chosen. The CVE ID may show a year value that does not match the release date, however, the release date will fall within the chosen year and month." It also mentions "1186 entries found for March 2019". A grid of CVE IDs is listed, with the first few rows being:

CVE-2019-30709	CVE-2018-20799	CVE-2018-8790	CVE-2019-8278	CVE-2019-8279	CVE-2019-9482
CVE-2019-9483	CVE-2019-9484	CVE-2019-9543	CVE-2019-9544	CVE-2019-9545	CVE-2019-9546
CVE-2019-9547	CVE-2019-9549	CVE-2019-9550	CVE-2019-9551	CVE-2019-9552	CVE-2017-15515
CVE-2018-5482	CVE-2019-6206	CVE-2019-6235	CVE-2019-9563	CVE-2019-9565	CVE-2019-9566
CVE-2019-9567	CVE-2019-9568	CVE-2018-11793	CVE-2018-15361	CVE-2018-1875	CVE-2018-1899

<https://nvd.nist.gov/vuln/full-listing/2019/3>

MARCH ALONE 1186 NEW VULNERABILITIES REPORTED SO FAR

At the time of slide creation 1186 vulnerabilities just for the month of March 2019.



<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.

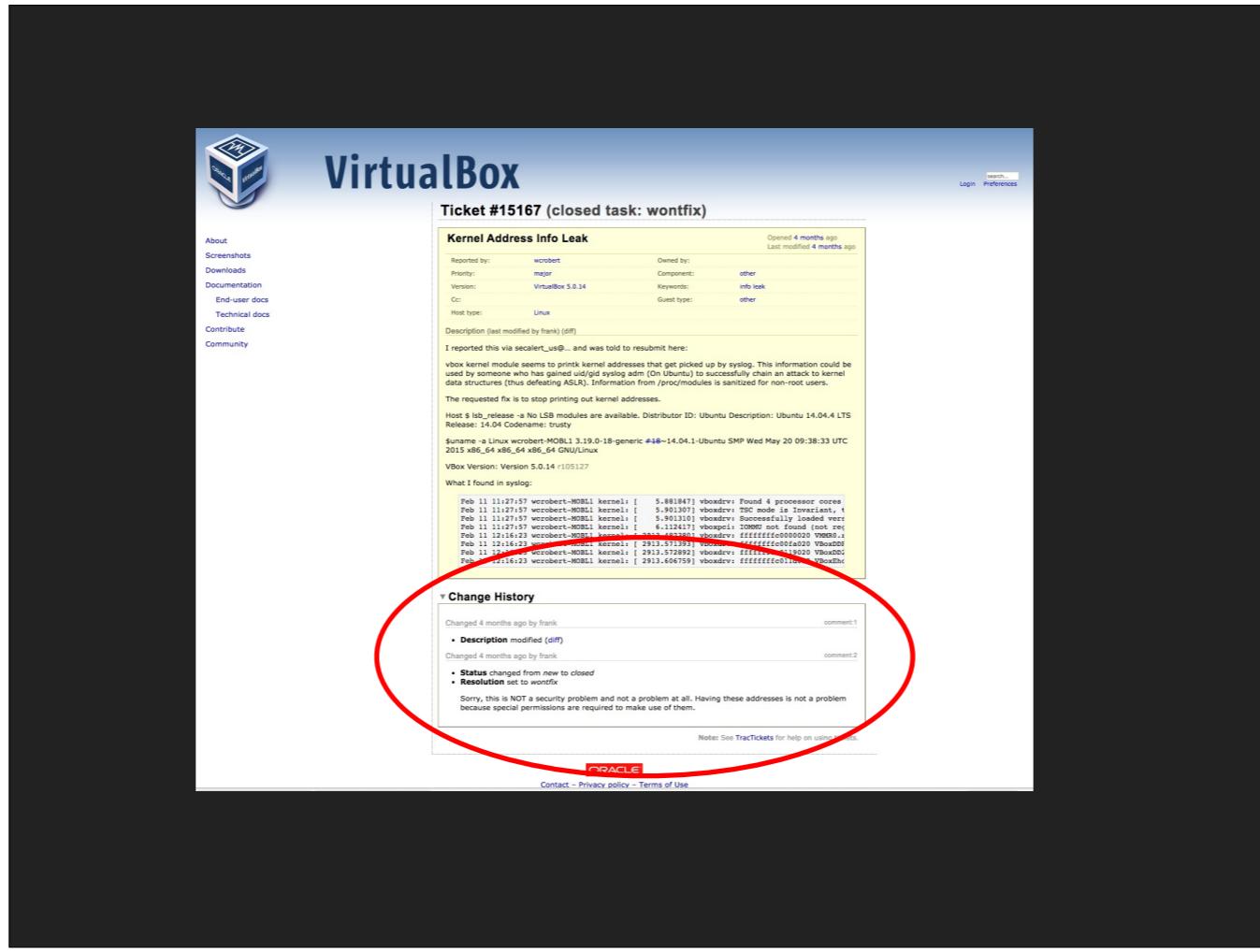
<https://www.cvedetails.com/browse-by-date.php>

- 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.
-

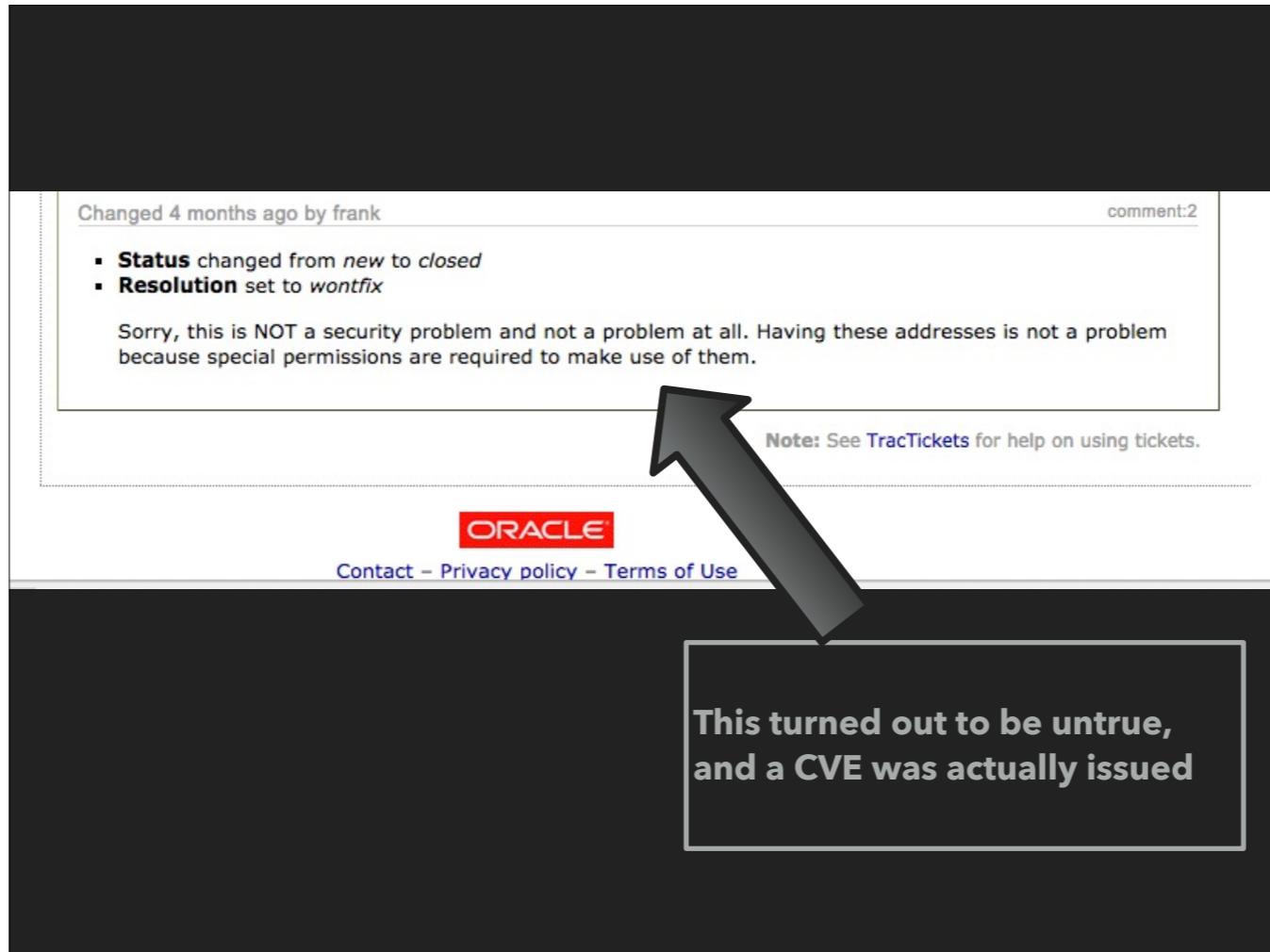
WHY?

Good security reporting does not guarantee action

And one last one to note is that good security reporting doesn't actually guarantee good action. You're probably not going to know which companies and groups have good reputations, and that's why you rely on the expertise of the open source group at Intel, but I want you to be aware that this is an issue that could come up.



Here's one that's come up recently. You probably can't read this on the screen, but this is a closed bug reported by one of our security team members, Bill Roberts. It is just an innocuous little leakage of kernel space address.



Zooming in so you can see it, you can see that the person who triaged this bug said it wasn't an issue. But thanks to the work of our intern, John Anderson, who provided additional information and a proof of concept, they've now acknowledged this as a real issue and are issuing a public vulnerability number for it and getting it fixed. But if he hadn't persisted, this security issue would have still been an issue in virtualbox indefinitely!

NOTE: There is always someone willing to spend the time to create an exploit. Whether for money or fame someone somewhere will do it just because it is there.

STEP 5: 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.

Test Plan:

**Run tests to make sure we're not
crazy, and cross fingers.**

- From <https://github.com/Khan/react-components/commit/539b0568b983d534b5c186b588a47ed462da75db>

IS THERE A 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

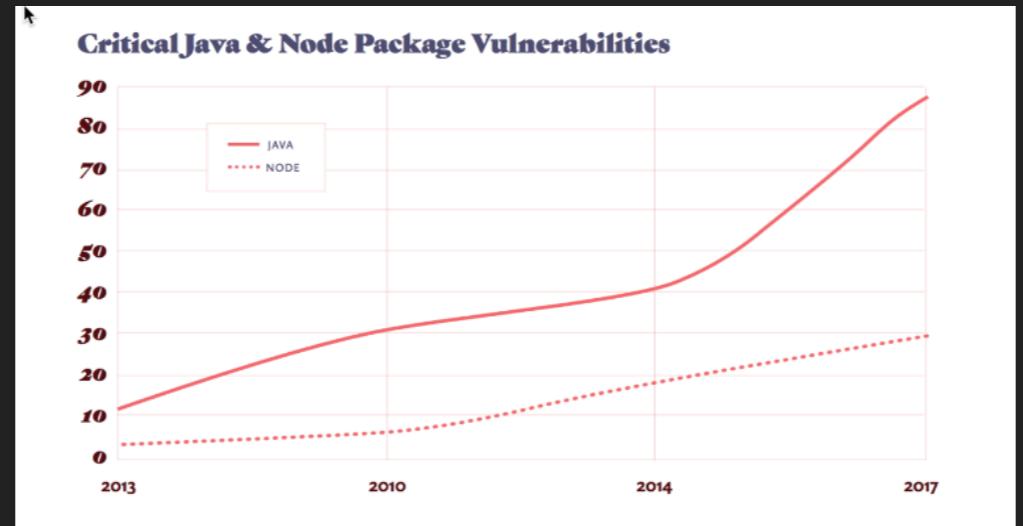
As before, here's some key questions for folk who aren't sure what a "good" test suite looks like. We're looking for a good variety of tests covering good and bad behavior, and test suites are really important for things that parse user input.

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.

Popularity does not equal security

First off, popularity doesn't equal security. Open source proponents like to talk about many eyes making all bugs shallow, but alas, many untrained eyes don't mean you're going to find finicky security bugs.



“90% of tested organizations use vulnerable dependencies.”

- <https://snyk.io/>

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.

SECTION	GRADE	GRADING GUIDELINES
Before you start	N/A	[1] Choose your repository wisely no forks unless you know it try to get the original maintained upstream source. [2] Check for compatible licenses
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 CI, results are published for builds. B - Project has test suite with good coverage but no CI 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.

CHECK THE LICENSE AND THE REPOSITORY
TAKE A LOOK (ARE THERE RED FLAGS?) REALLY READ IT
CHECK FOR THE NUMBER OF CONTRIBUTORS & ACTIVITY
BE AWARE OF YOUR DEPENDENCIES
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 vet everything with a some amount of scrutiny you will lower the projects overall risk exposure.

Thank you



LinkedIn: <https://www.linkedin.com/in/mikide/>

Twitter: [@theDawgCr8](https://twitter.com/theDawgCr8)

Email: sec-princess@unroutable.me

Intel Credits: Terri Oda, Tiberius Heflin, Bill Roberts, John Anderson

<https://github.com/sec-princess/F5-SecurityJam>

LinkedIn: <https://www.linkedin.com/in/mikide/>

Twitter: [@theDawgCr8](https://twitter.com/theDawgCr8)

Email: sec-princess@unroutable.me

<https://github.com/sec-princess/WWCode-OSS-Study-Night-20180927>