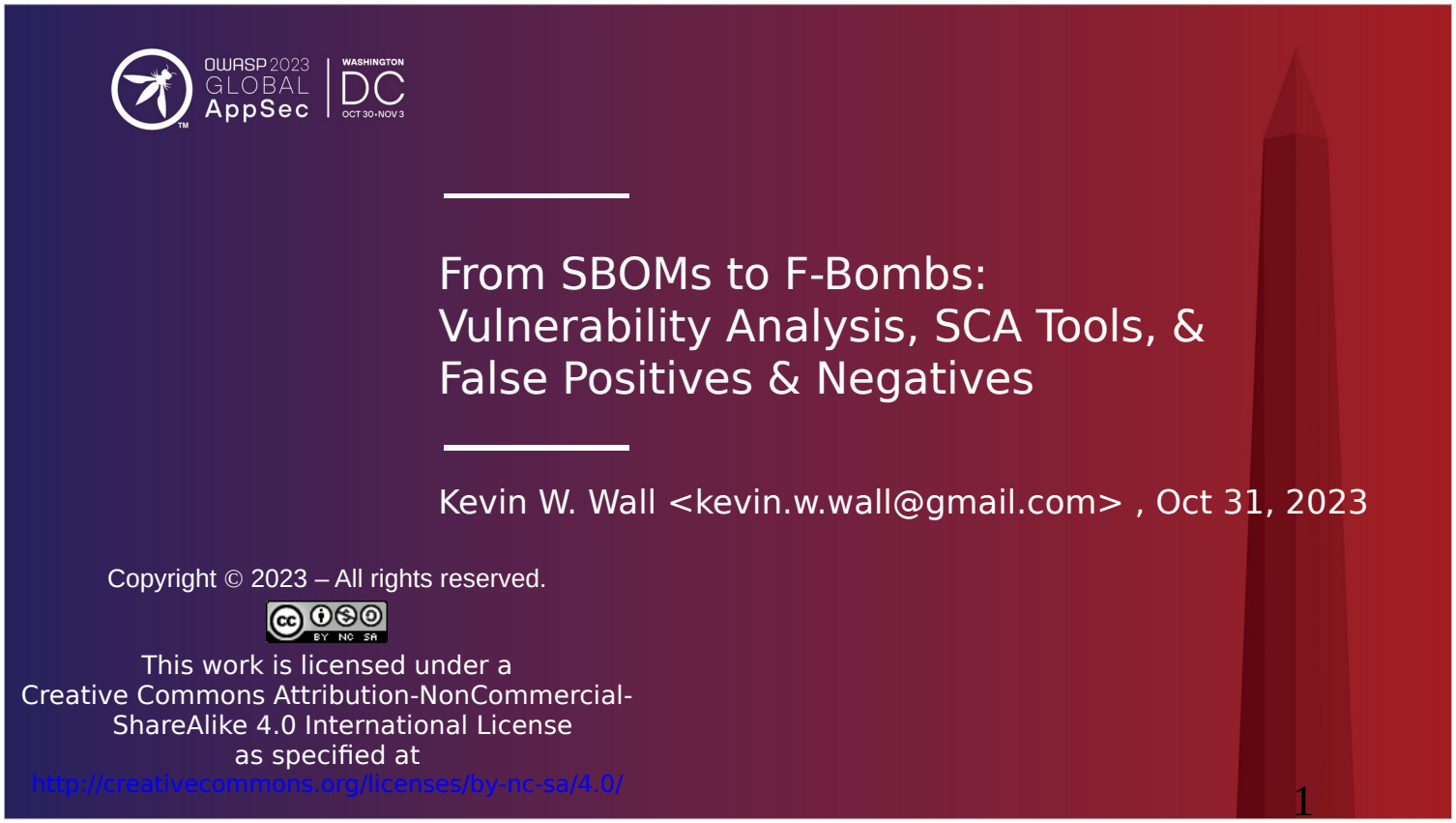


Hello, and welcome to my talk. After seeing my title, I want to ensure you that this talk is rated PG and I'm not planning to be dropping any F-bombs. Although I must confess, addressing the patching of the vulnerabilities in Apache Commons FileUpload dependencies that resulted on my language most definitely being R-rated, which is what lead me to that title in the first place.

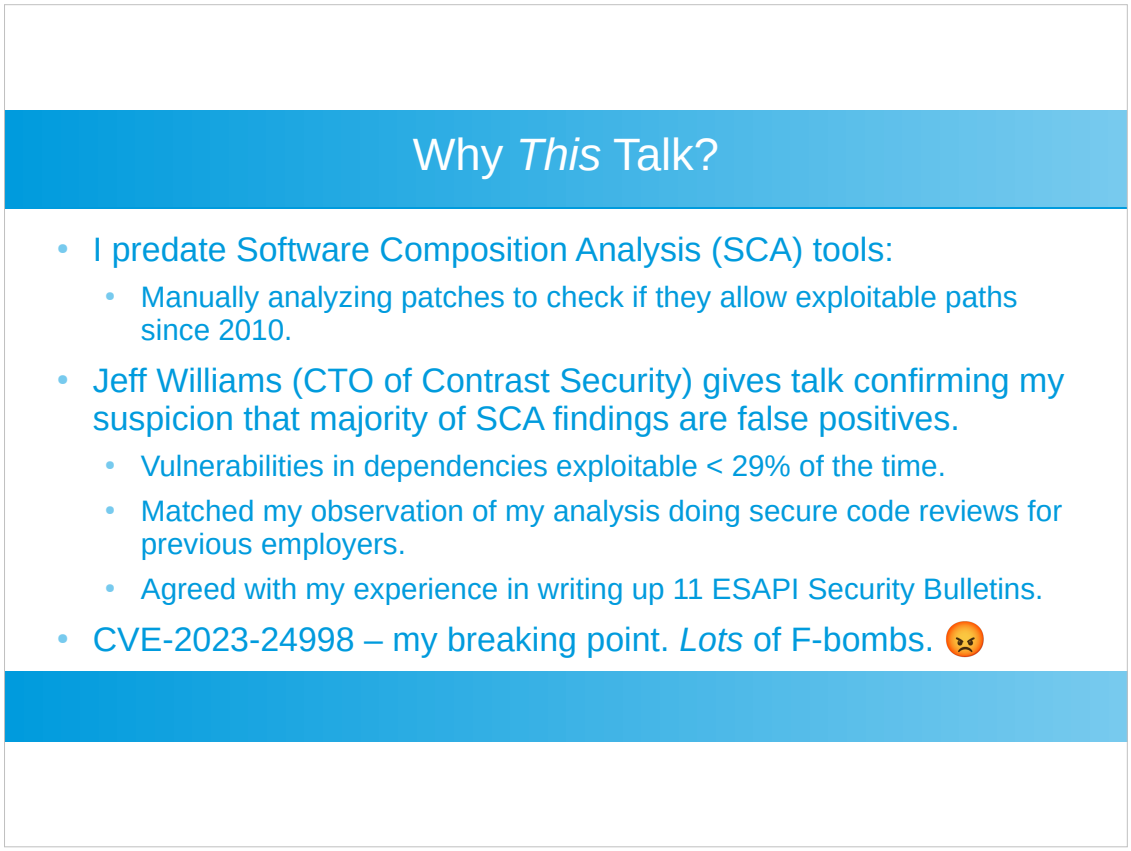
Also, just so I don't get censored by OWASP or my company or any colleagues, I need to say that the views expressed in this talk are mine alone and not those of my employer or the OWASP Foundation. Also, I don't necessarily expect most of you to agree with them as several of the expressed opinions are definitely not part of the traditional AppSec party line.

Who is This Knucklehead?

- 20+ years systems programming, 20+ years in AppSec.
- Previously developed several *proprietary* systems libraries, including one for AppSec.
- OWASP ESAPI involvement (ESAPI dates back to 9/2007)
- Involved in ESAPI since June, 2009.
- Redesigned and re-implemented symmetric encryption
- Have been ESAPI co-lead since 2011.
- Buzzword free talk. No mention of ChatGPT or quantum computing! (Unless you consider SBOM a buzzword! :)

- My analysis experience goes back to the days of the obtuse “multiple undisclosed vulnerabilities” that Oracle was famous for (or should I say infamous?) when describing what they updated in their security releases. I learned back then to reverse engineer the patched jar files and then provide details to my InfoSec management to help them decide how critical it was to patch immediately or whether it could be deferred for a while.
- Actual number for vulnerabilities in dependencies probably is less than 10%. Williams claims:
  - 62% of time, dependent libraries never loaded at all!
  - Only 9% of running application is 3rd party libraries.
- CVE-2023-24998 addressed a DoS vulnerability in Apache Commons FileUpload.

Why the F-bombs? 1) The CVSS was 7.5, which is nuts for DoS, and 2) in my opinion, they screwed up the patch in multiple ways. It’s a Band-Aid that really doesn’t do much. We’ll revisit that part of the CVE fix later.



- My analysis experience goes back to the days of the obtuse “multiple undisclosed vulnerabilities” that Oracle was famous for (or should I say infamous?) when describing what they updated in their security releases. I learned back then to reverse engineer the patched jar files and then provide details to my InfoSec management to help them decide how critical it was to patch immediately or whether it could be deferred for a while.
- Actual number for vulnerabilities in dependencies probably is less than 10%. Williams claims:
  - 62% of time, dependent libraries never loaded at all!
  - Only 9% of running application is 3rd party libraries.
- CVE-2023-24998 addressed a DoS vulnerability in Apache Commons FileUpload.

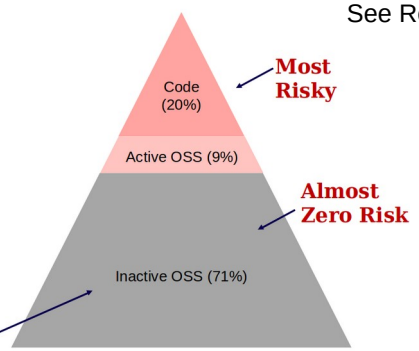
Why the F-bombs? 1) The CVSS was 7.5, which is nuts for DoS, and 2) in my opinion, they screwed up the patch in multiple ways. It’s a Band-Aid that really doesn’t do much. We’ll revisit that part of the CVE fix later.

SCA Bandages for Open Sores Security?

Open Source Security

- **Average app/API has...**
  (based on 100K+ real world apps/APIs)
  - **9% active library code**
  - **71% inactive library code (never loaded, never invoked)**
  - **62% of libraries are totally inactive with zero active code**
  - **Most of the risk comes from the 20% custom code**
- **"Vulnerabilities" reported in inactive code are false positives**

See References for link.

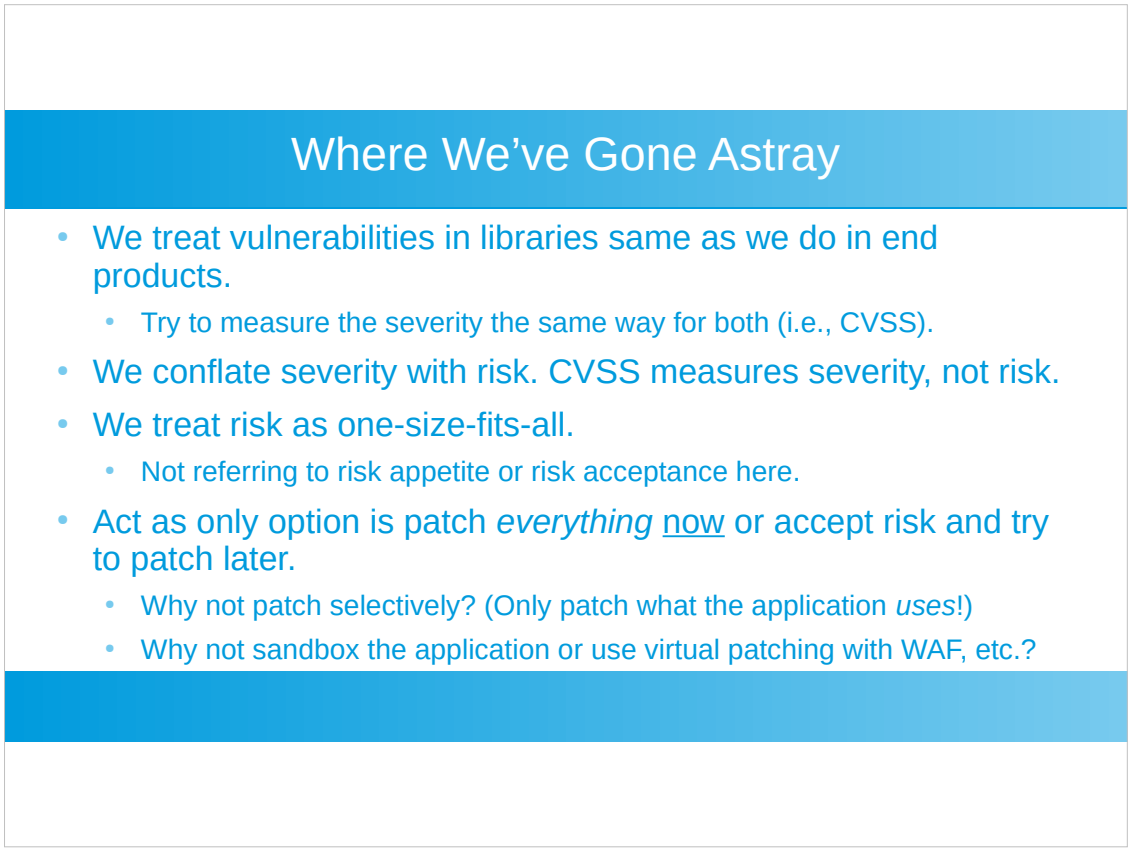


* Contrast State of Open Source (OSS) Security Report 2021

Contrast

This slide content is from Jeff Williams, CTO of Contrast Security. (I won't blame him for the bad pun on this slide title however; that's all my doing.)
However, given all the false positives that static SCA tools produce, I thought it was appropriate,

The reason why I've included this slide is as a source of the 2nd bullet from the previous slide. It is based on lots of data collected by Contrast Security.



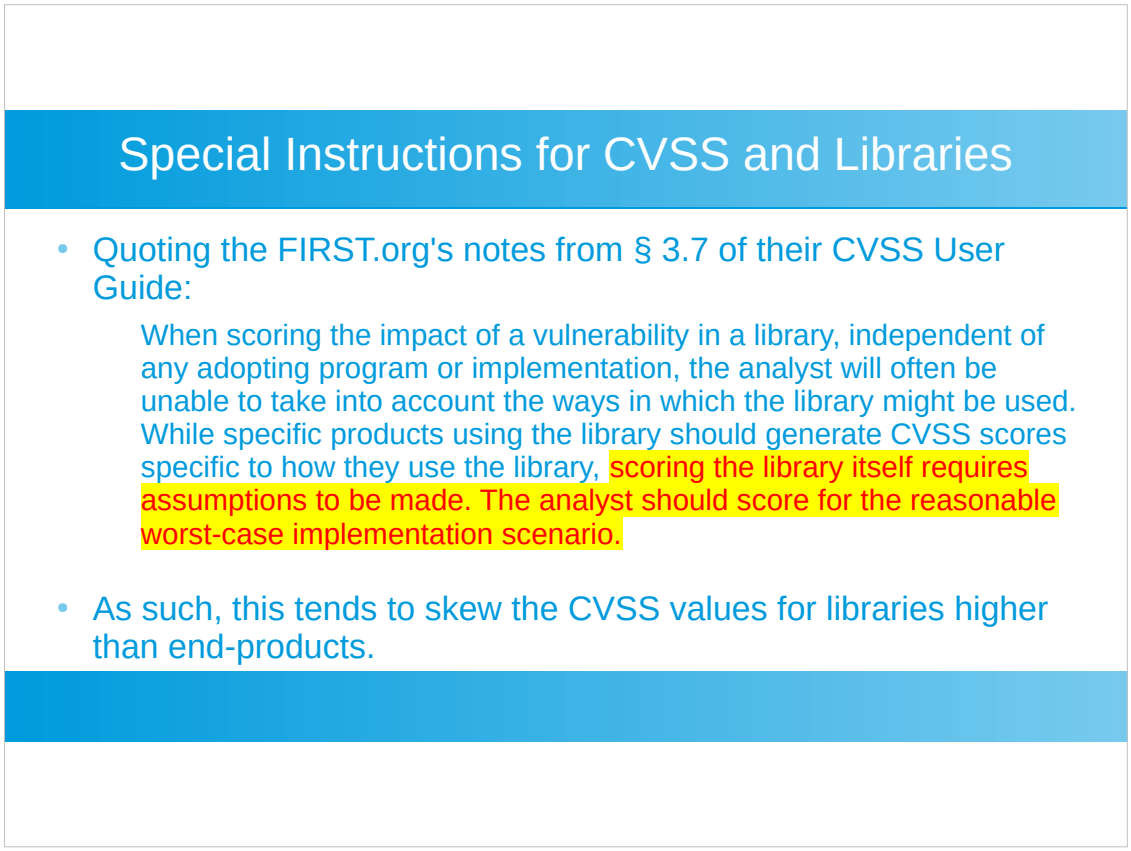
As as a culture, humans tend to overreact to new risks. And while SCA tools were a thing before Log4Shell (you all remember OWASP Dependency Check, right?) that was the watershed moment that changed everything. Most companies were unprepared for the fallout. But it was "lesson learned", right? The mantra became "never again", so we all started scanning dependencies with a vengeance and insisting that *everything* be patched, especially vulnerabilities that were High.

- So how did we determine what a "High" risk vulnerability was? Simple, we believed whatever the SCA scores told us and most of them just used the CVSSv3 base scores from NVD, which is not without problems.
- We ignored impact to development teams and that, in fact, most of the vulnerabilities in dependencies were not exploitable in our applications.
- In doing this, the pendulum swung from one extreme to the other.

[First: Highlight bullet points, then…] So let's take a quick detour and compare CVSS base sores on 2 different CVEs. The next slide will highlight:

- CVE-2023-24998 – DoS in Apache Commons FileUpload; CVSSv3 – 7.5 (High)
- CVE-2023-0669 – Pre-AuthN remote command injection in the product Fortra GoAnywhere MFT. Discovered in 0day attacks; CVSSv3 – 7.2 (High)

Does it make sense to you that a DoS vulnerability in a library that has no evidence of every being actually exploited scores higher than an RCE vulnerability in a product that was subject of multiple 0day attacks? Intuitively, no. But if you look at the CVSS vector string for the Commons FileUpload vulnerability, you will see it was calculated using an absolute worst case scenario as per the First.org's CVSS User Guide.



Emphasis is mine here.

Now I understand why this is recommended. I think it's at least partly because these CVSS scores are used as a severity to either compute risk ratings or, as I previously mentioned, often incorrectly just treated as a risk rating itself. So, I'm sure that FIRST did not want the risk calculations to be vastly underestimated. (CVSS 1.0 was released in Feb 2005, and the 9/11 attacks were still fresh in people's minds.) So they set up this worst case scenario as guard rails. I also think it's easier to reason about worst case scenarios than average case ones.

But the worst case scenario by comparison also rarely happens.

Detour: Comparing CVSSv3 scores

- CVE-2023-24998 – DoS in Apache Commons FileUpload
  - Type: Library
  - Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H
  - Base score: 7.5 (High)
- CVE-2023-0669 – Pre-Authentication RCE in product Fortra GoAnywhere MFT
  - Type: Product
  - Vector: CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:U/C:H/I:H/A:H
  - Base Score: 7.2 (High)

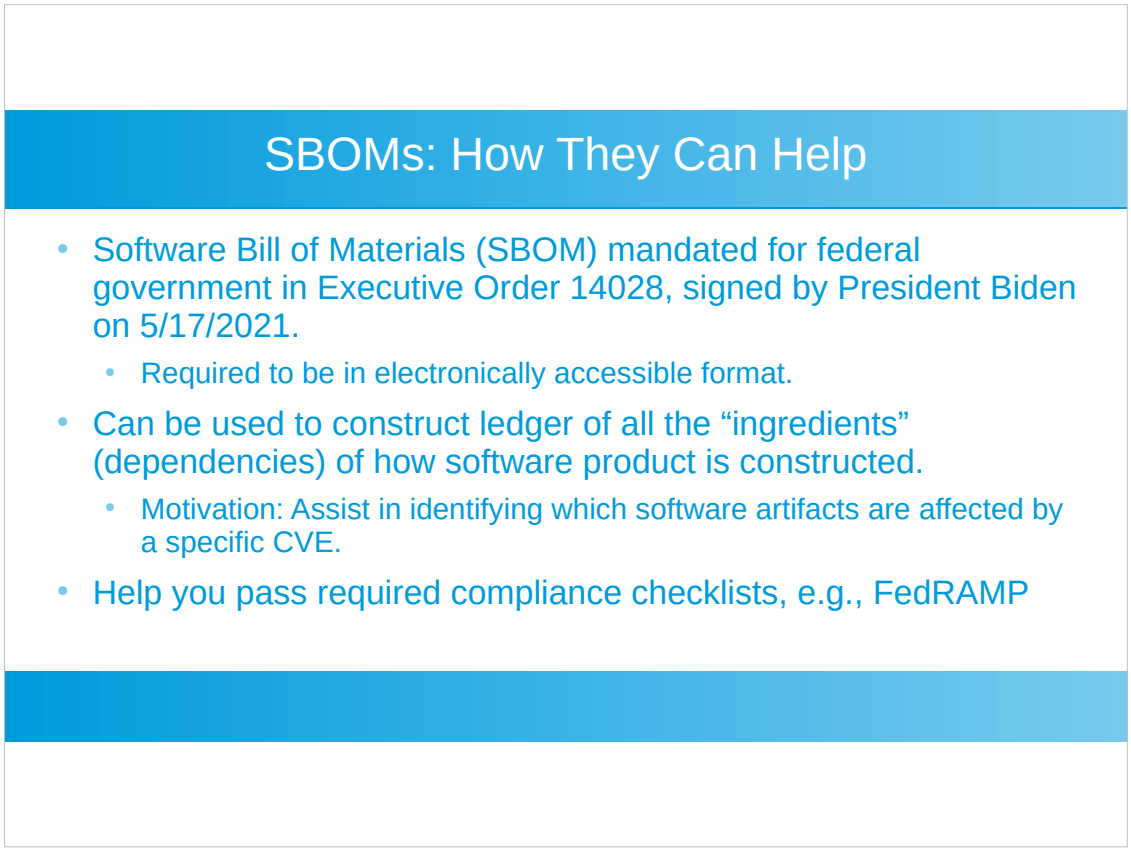
PR is “Privileges Required”; ‘N’ means ‘None’ and ‘H’ means ‘High’.

A few things to note here. First off, NIST is ***not*** biased here. They scored both of these by the book, using the worst case assumptions as noted on the previous slide.

Secondly, the Forta CVE had 0day attacks observed in the wild.

The ‘Privileges Required’ being ‘High’ here significantly lowers the overall CVSSv3.1 base score. Had it been a ‘Low’ instead of a ‘High’, the base score would have been 8.8 rather than 7.2. And had it been ‘None’ the way it was for the Apache Commons FileUpload, it would have resulted in a base score of 9.8 (Critical), all other things being equal. So CVSS is telling us here is that ease of exploit is very important to the overall score. (But working 0day exploits, eh?)

Lastly, CVSS also includes a Temporal Score and an Environmental Score, neither of which NVD includes, but may very be important to your organization.



These dependency “ingredients” usually have a unique software component name, a package URL repository of where it was downloaded from, the file type, a human readable version of the dependency, and various message digests (i.e., secure hashes) presenting a message integrity code of the dependency. (The secure hashes include MD5 and SHA1 [both known to be broken] as well as SHA-256, SHA-384, and SHA-512.)

The idea if one of these lists of “ingredients” are later found to contain some particular “poison” [air-quotes], that is, a known *unpatched* vulnerability, then we can later discover which software products are affected by searching all of our software artifacts using something like Dependency Tracker, to tell which products need to have their libraries updated (i.e., a new updated, patched version).

This also would work with vulnerabilities yet to be discovered.

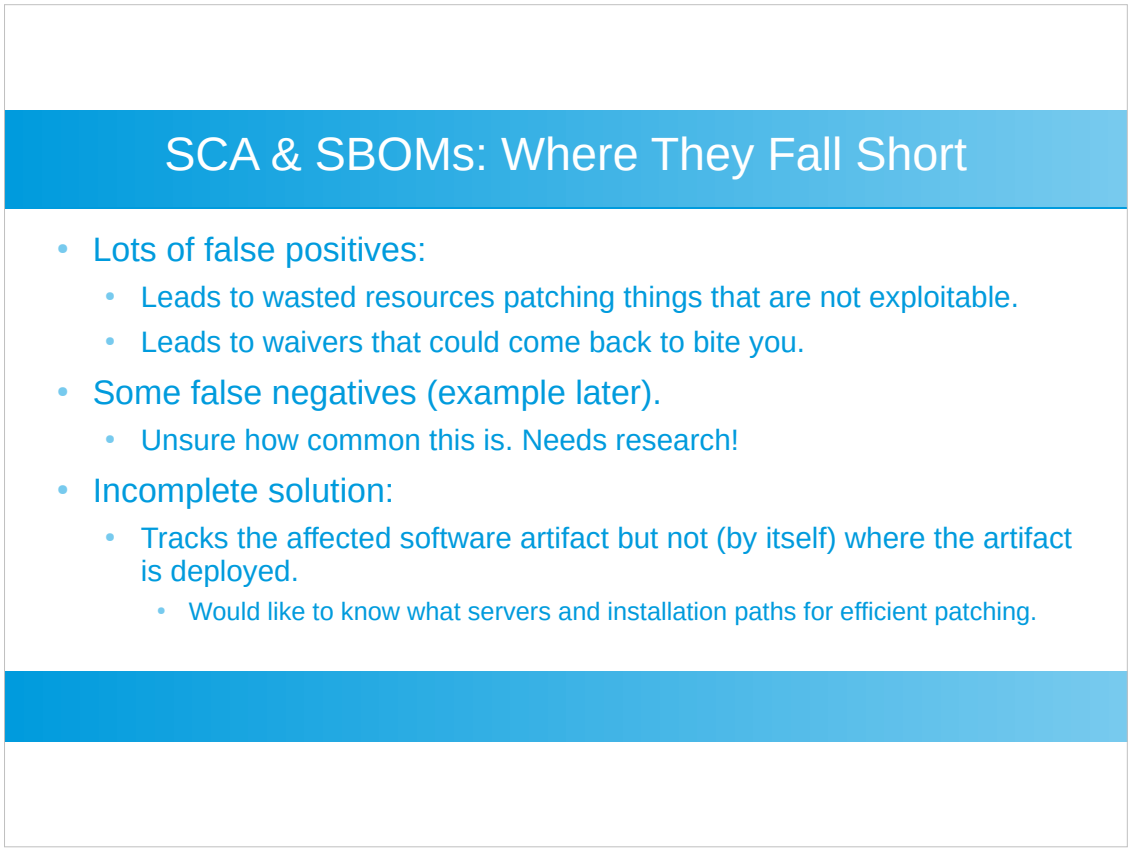
SBOM Promise: What We Expect Them To Do

> “The idea behind such a thorough inventory is that companies can better track the nuts and bolts of their software—including whether it houses security vulnerabilities like the Log4j software flaw—and more quickly respond to them.”
>
> — *The Wall Street Journal*, “AI Is Generating Security Risks Faster Than Companies Can Keep Up”, 2023-08-10, https://www.wsj.com/articles/ai-is-generating-security-risks-faster-than-companies-can-keep-up-a2bdedd4

I recently ran across this statement in an article about AI and how—in the short term at least—it is causing more security problems than it is solving. This particular blurb was in the context of SBOMs, and they are not wrong. Can we all admit that this is our hope and it’s how we sold our management on our shiny new Software Composition Analysis and/or SBOM tool of choice?

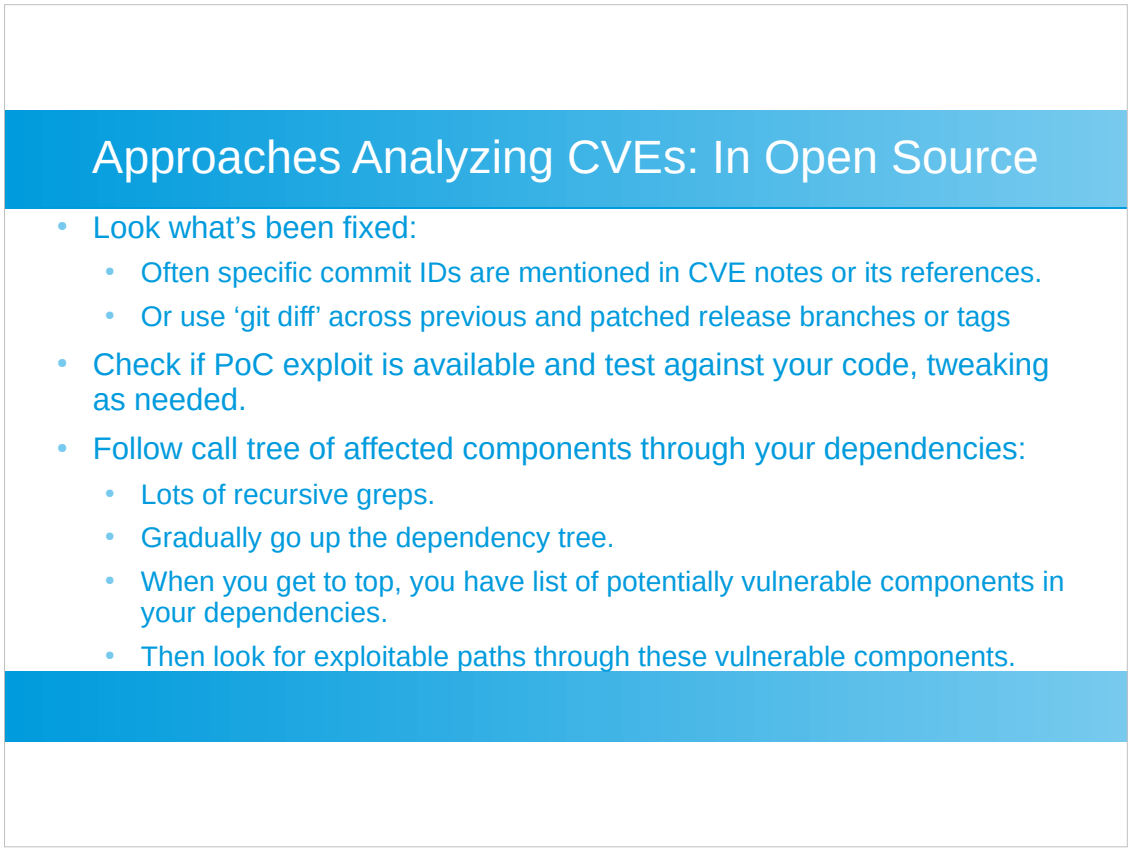
In fact, I think this likely unfulfilled promise is the major reason why SBOMs made their way into President Biden’s Executive Order 14028.

The promise is partly empty though, because there’s a major underlying faulty assumption made by SCA tools and conveyed by SBOMs. I’ll talk about that next.



The big problem if you have a few thousand software projects, is keeping everything completely patched according to some regular patch schedule. That is extremely difficult. This is because:

- The list of dependencies (direct + transitive) is often huge even on a per project basis. Over a few hundred per application seems typical.
- There typically are 20k+ new CVEs per year, so it's hard to keep up to say nothing of those from private vulnerability databases.
- Library owners don't deploy patched releases as quickly or as regularly as we'd like.
- Sometimes patching will break things or devs fear they may break things because of inadequate regression tests.
- As a result, over time, an enormous backlog of dependency patching results.
- However, that pyramid slide shows us that ~62% of those libraries never even get loaded, which means that most vulnerabilities reported in dependencies are not even exploitable because the vulnerable code is never loaded! That is, they are false positives.

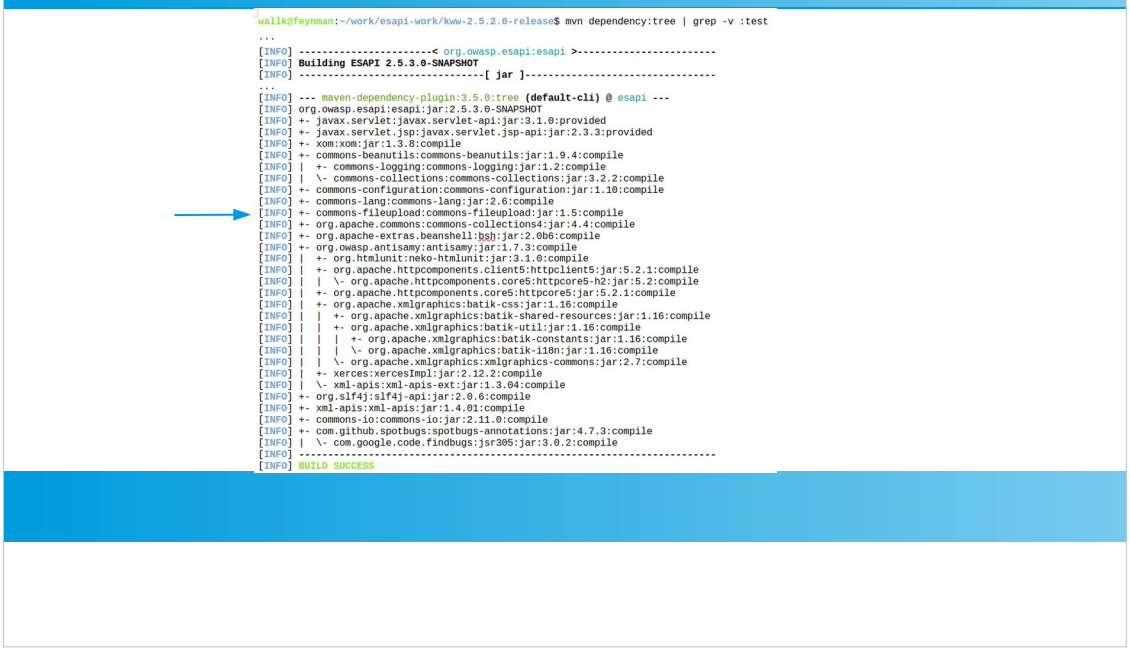


Sometimes identifying what has been fixed is a huge challenge.

Ever see a description of a patch that says something like "Multiple undisclosed vulnerabilities were patched in several classes"? Yeah, you've seen it because for years shortly after Oracle acquired Sun Microsystems, that's how Oracle used to describe the patches in their release notes.

- At least back then you could extract the jar and generally tell what had been changed based on the modification dates of the .class files in the jar (typically, Java's rt.jar).
- But thanks to SBOMs (to make reproducible hashes), those random modification timestamps are pretty much gone.
- Fortunately OpenJDK at least reports details, so there's that.

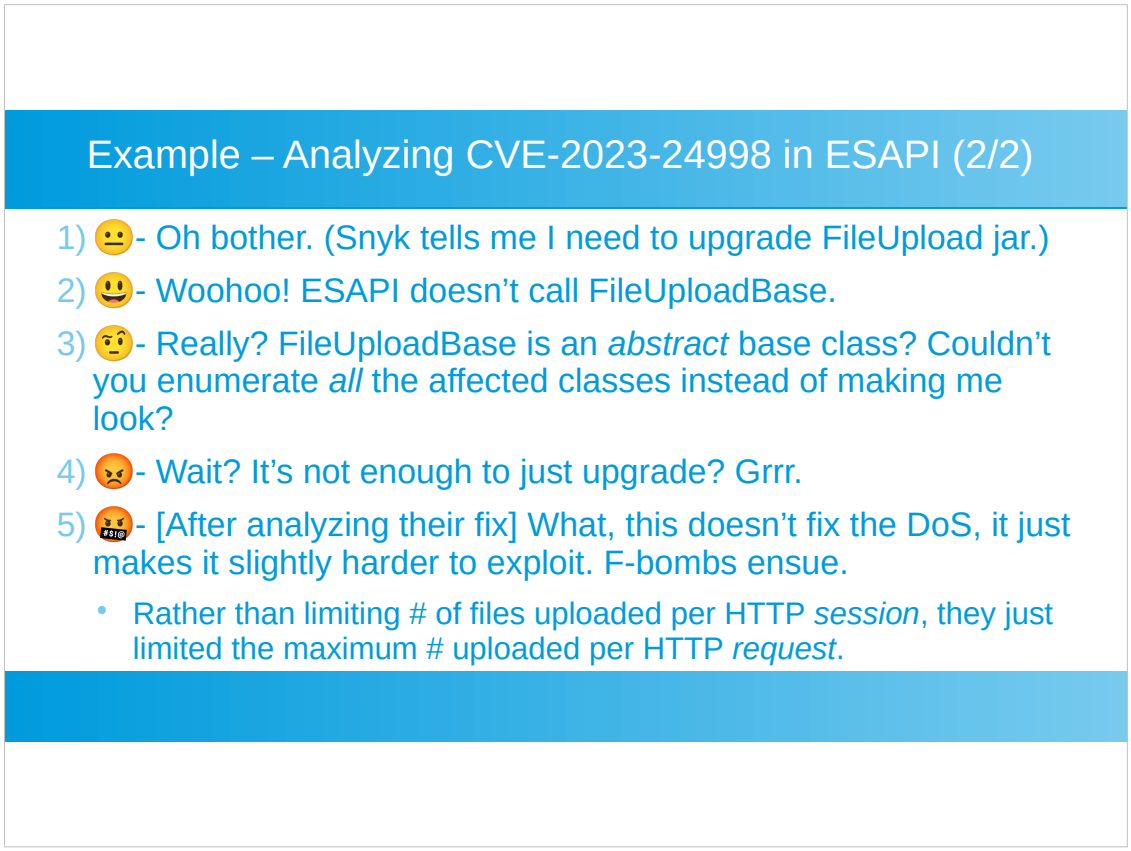
Example – Analyzing CVE-2023-24998 in ESAPI (1/2)



```
wallk@feynman:~/work/esapi-work/kww-2.5.2.0-release$ mvn dependency:tree | grep -v :test
...
[INFO] ----------------------< org.owasp.esapi:esapi >-----------------------
[INFO] Building ESAPI 2.5.3.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
...
[INFO] --- maven-dependency-plugin:3.5.0:tree (default-cli) @ esapi ---
[INFO] org.owasp.esapi:esapi:jar:2.5.3.0-SNAPSHOT
[INFO] +- javax.servlet:javax.servlet-api:jar:3.1.0:provided
[INFO] +- javax.servlet.jsp:javax.servlet.jsp-api:jar:2.3.3:provided
[INFO] +- xom:xom:jar:1.3.8:compile
[INFO] +- commons-beanutils:commons-beanutils:jar:1.9.4:compile
[INFO] |  +- commons-logging:commons-logging:jar:1.2:compile
[INFO] |  \- commons-collections:commons-collections:jar:3.2.2:compile
[INFO] +- commons-configuration:commons-configuration:jar:1.10:compile
[INFO] +- commons-lang:commons-lang:jar:2.6:compile
[INFO] +- commons-fileupload:commons-fileupload:jar:1.5:compile
[INFO] +- org.apache.commons:commons-collections4:jar:4.4:compile
[INFO] +- org.apache-extras.beanshell:bsh:jar:2.0b6:compile
[INFO] +- org.owasp.antisamy:antisamy:jar:1.7.3:compile
[INFO] |  +- org.htmlunit:neko-htmlunit:jar:3.1.0:compile
[INFO] |  +- org.apache.httpcomponents.client5:httpclient5:jar:5.2.1:compile
[INFO] |  |  \- org.apache.httpcomponents.core5:httpcore5-h2:jar:5.2:compile
[INFO] |  +- org.apache.httpcomponents.core5:httpcore5:jar:5.2.1:compile
[INFO] |  +- org.apache.xmlgraphics:batik-css:jar:1.16:compile
[INFO] |  |  +- org.apache.xmlgraphics:batik-shared-resources:jar:1.16:compile
[INFO] |  |  +- org.apache.xmlgraphics:batik-util:jar:1.16:compile
[INFO] |  |  |  +- org.apache.xmlgraphics:batik-constants:jar:1.16:compile
[INFO] |  |  |  \- org.apache.xmlgraphics:batik-i18n:jar:1.16:compile
[INFO] |  |  \- org.apache.xmlgraphics:xmlgraphics-commons:jar:2.7:compile
[INFO] |  +- xerces:xercesImpl:jar:2.12.2:compile
[INFO] |  \- xml-apis:xml-apis-ext:jar:1.3.04:compile
[INFO] +- org.slf4j:slf4j-api:jar:2.0.6:compile
[INFO] +- xml-apis:xml-apis:jar:1.4.01:compile
[INFO] +- commons-io:commons-io:jar:2.11.0:compile
[INFO] +- com.github.spotbugs:spotbugs-annotations:jar:4.7.3:compile
[INFO] |  \- com.google.code.findbugs:jsr305:jar:3.0.2:compile
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

General steps involved are:

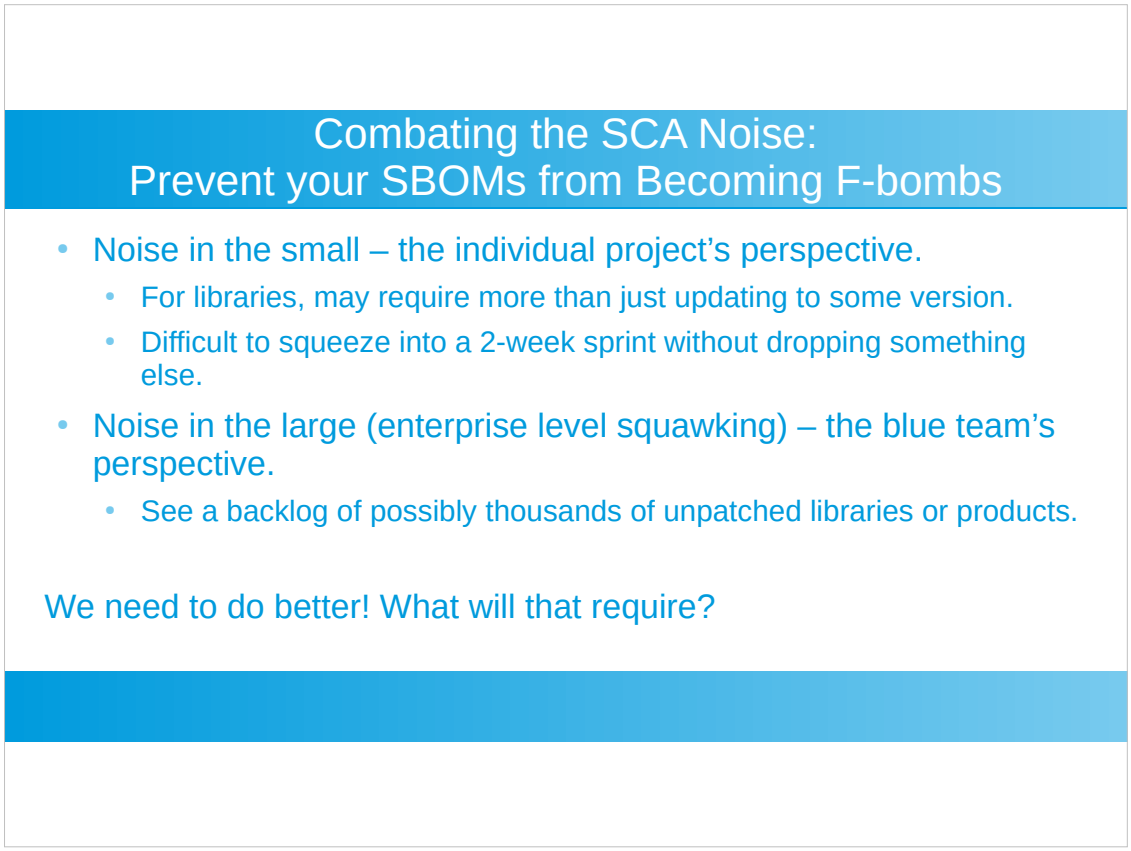
1) Identify library / version currently in use and download its source.
2) If library is direct dependency, done. Otherwise, work your way up the transitive dependency tree to your application code and download source code of appropriate dependency version for each.
3) Start recursively grepping for affected methods or classes in library associated with CVE of interest.
   a) Don't forget to check configuration files for reflection use.
   b) Beware interfaces and subclasses!
4) Recursively grep next library level up for all previous results (methods and/or classes).
5) Continue process until arriving at your application or library.
6) Starting at your application or library, draw a reachability tree to trace possible exploitable paths.
7) Do deep dive of all possible paths to see if any exploitable ones exist.
8) If exploitable path(s) exist, write up analysis notes.



My descent from blessed to cursed, in 5 steps. [Read steps; pause at #3 to note below.]

Step 3: Remember where I mentioned "Beware interfaces and subclasses" on the previous slide? Yeah, that could have bit me here, but writing up Security Bulletin #11 for ESAPI required a much deeper analysis. I usually write those up even if ESAPI is not affected to describe workarounds to suppress the blaring SCA alarms.

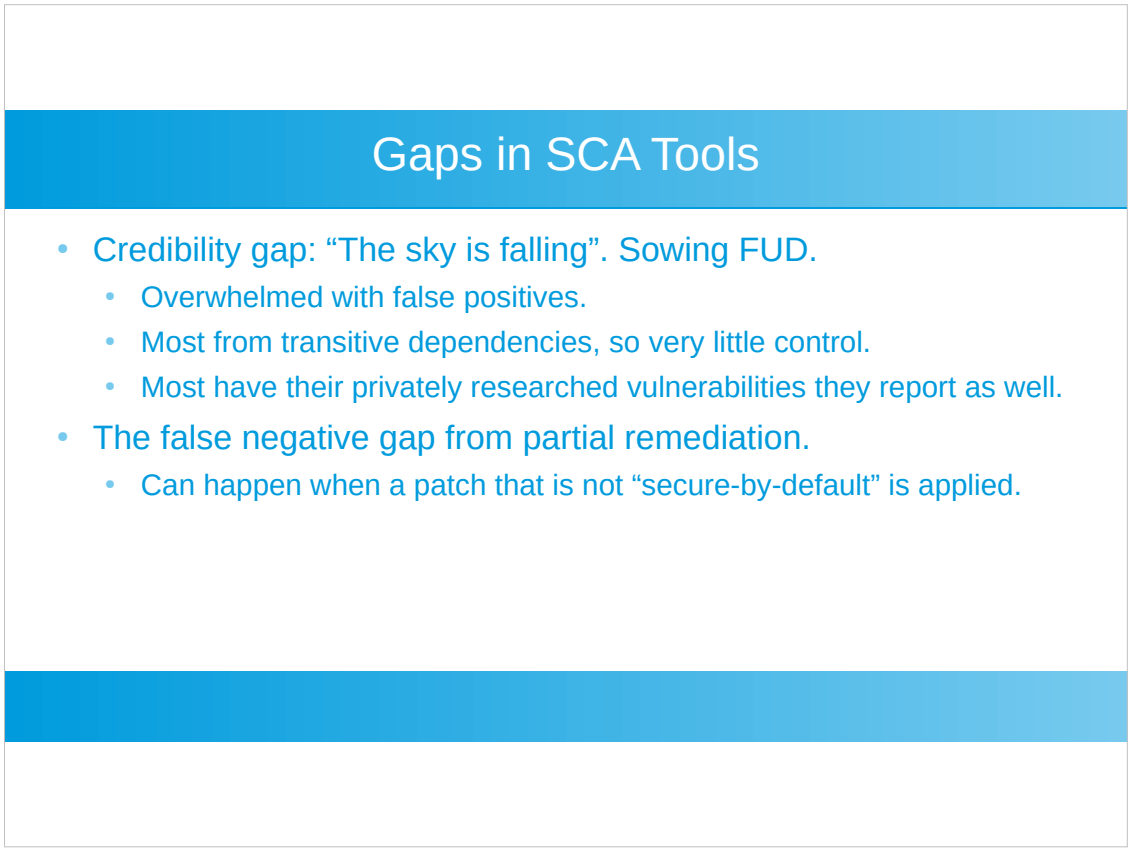
However, as it turned out in this case, ESAPI was calling Apache Commons FileUpload's ServletFileUpload class which extends their FileUpload class which in turn extends that abstract class FileUploadBase. So ESAPI *was* affected.



For *direct* dependencies at least, developers can usually easily figure out if their application is vulnerable, but even when it's not, InfoSec still tells them that the "must patch" regardless (unless, perhaps patch is only in the next 'major' version). But this analysis is much harder for transitive dependencies and patching is potentially riskier. Could break things, (What's percentage code coverage in your regression tests???)

From your blue team's PoV, the best way to make the problem disappear to to force patching of everything and this is often expected in cadence with their normal OS patching or product release.

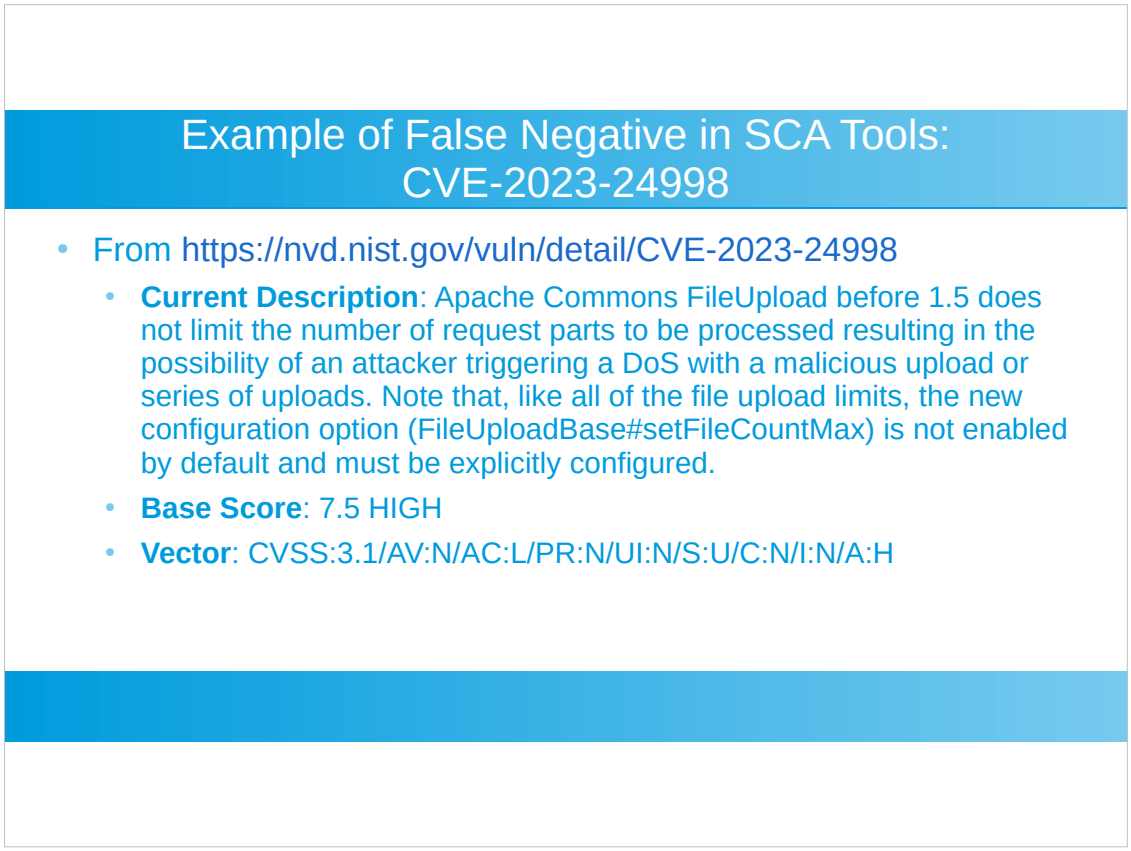
- OS vulnerability patching is more mature and **centralized**.
- By comparison, regular vulnerability patching *application* dependencies is only a decade old at best (or since Dec, 2019 for those who waited for the Log4Shell debacle to react).



I personally find the private vulnerability databases of the SCA vendor tools an anathema It’s like they are using that as a competitive advantage to the disadvantage of their customers and ultimately, the whole FOSS user community.

- There is little to no transparency regarding the details. Specifically, there is no way to vet their research.
- There’s reason to believe that the major SCA tool vendors have a large commonality across their private vulnerability DBs, but there are also likely lots of differences. Do they think we are going to license an SCA tool from each vendor?
- A least one of them also reports *retracted* CVEs. Ran across one of those that the CNA retracted from AntiSamy but was still in a private vulnerability DB.

On the plus side, both Snyk and Sonatype’s NexusIQ are starting to provide experimental “reachability” metrics to help rule out false positives. Perhaps others are as well.

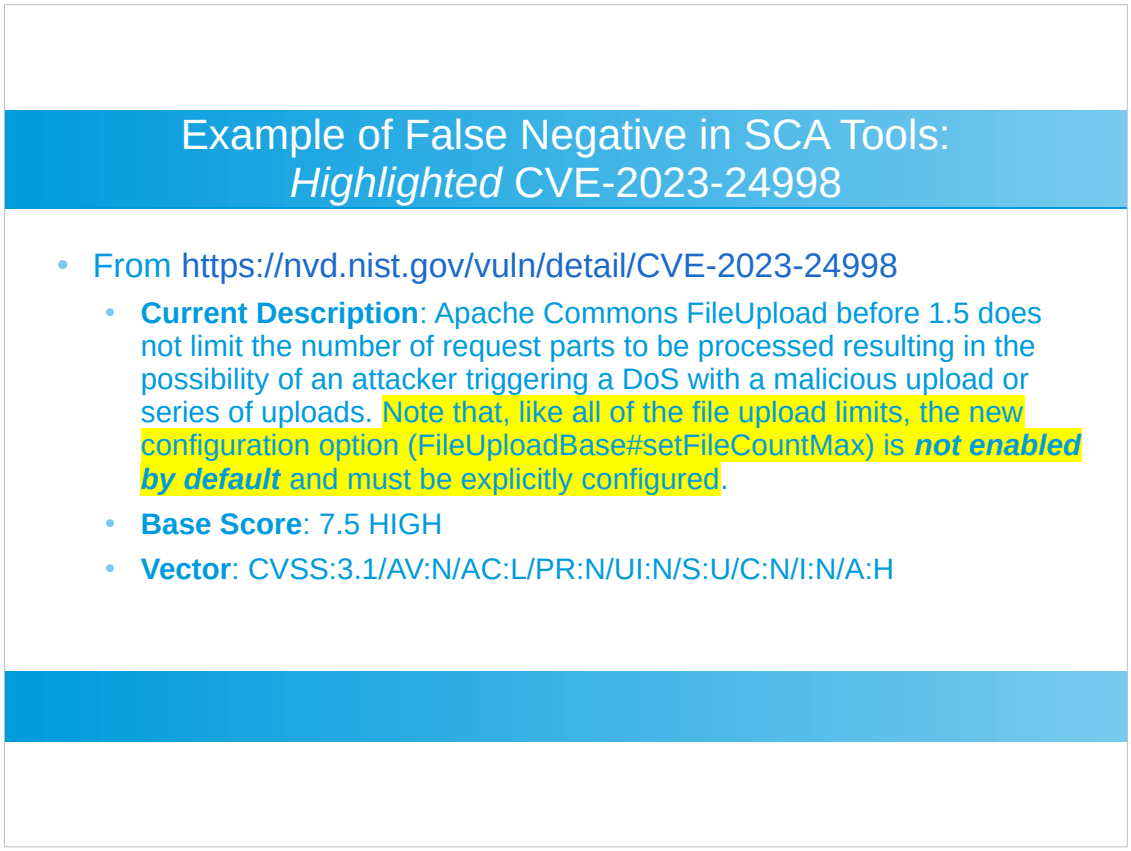


This CVE, by the way, was the source of all of my F-bombs! There are several issues with it.

For those of you in the back of the room, let me read the NVD Description of this CVE. Listen to it and see if you recognize the problem.

I'll give you a few seconds to ponder it. I'm sure if your looked at it long enough, it would jump out at you.

So as a developer, my first problem is that the FileUploadBase class that it refers to is an abstract base class, so if you grep your code to see if you are affected, are not going to find it. That was the cause of my first expletive. C'mon Apache: Help a fellow developer out. Show me all of the potentially affected concrete classes in your CVE description.

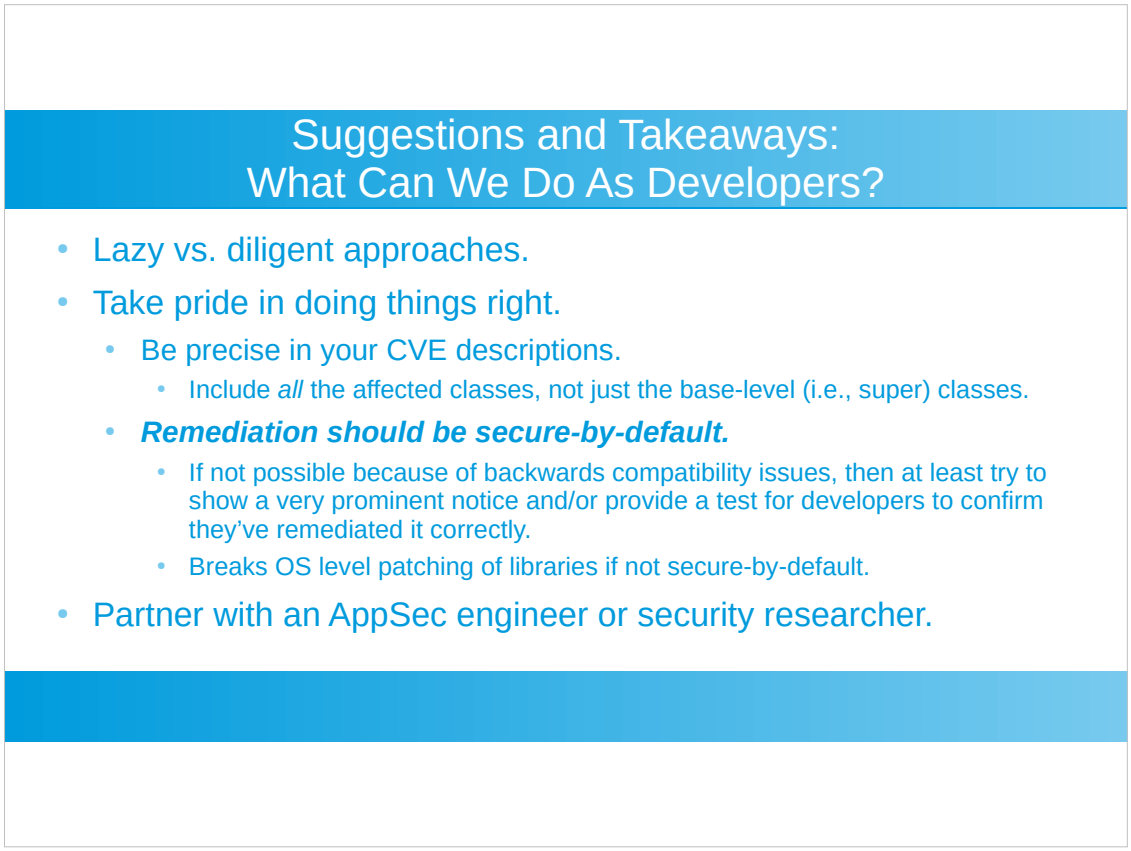


Look closer at the highlighted area. (Read again, out loud.)

This patch is **NOT** secure by default. What does that mean exactly?

Well suppose you are using something like Dependabot or Renovate or Snyk to watch for vulnerabilities in unpatched dependencies in your applications and you have it create PRs for your repo to update to the patched version. You merge the patch (possibly automatically if your workflows all pass). But you are STILL VULNERABLE because these tools will not add a call to setFileCountMax() at appropriate places in your code.

Furthermore, the SCA tools only are looking at dependencies from your pom.xml or build.gradle, etc. They have no clue that anything else needs to be done. *They are assuming patching makes it secure by default!*



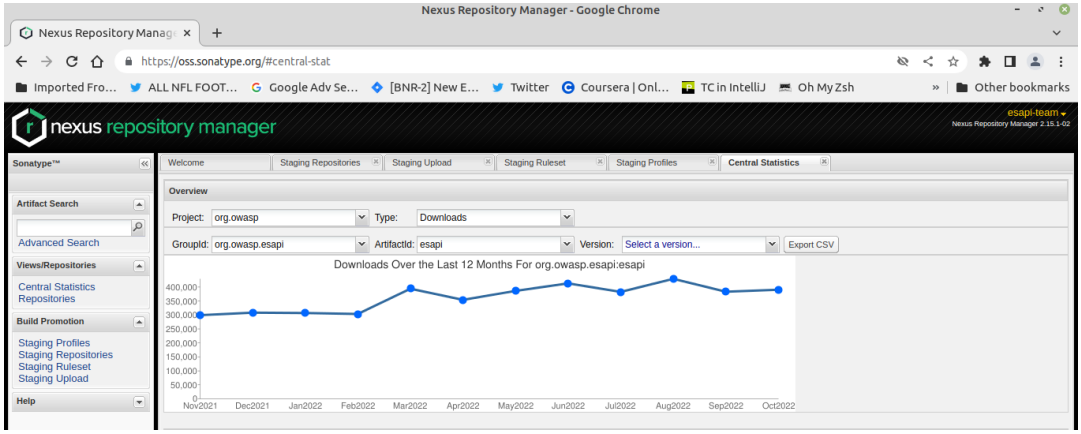
If you are a developer of a security library and you aren't responsive to patching vulnerabilities in your own library, then stop calling yourself a security library.

Also, if you are a security library or a library in the (say) top 10k libraries, you need to keep your library updated *even if it is not exploitable*.

- If this is not possible, you need to provide a detailed security bulletin to explain the situation.
- Looking at you OpenSAML. In top 2200 in MvnRepo, but last release (2.6.4) was in 2015 and still uses ESAPI 2.0.1, which was released on 7/25/2011! 2.0.1 wasn't even the newest release in 2015.

The next two slides use ESAPI as an example to show what I think is an indication of how badly we are failing to drive home this point. (Of course, ESAPI could just be an anomaly.)

Evidence That Many Are Not Patching (1/2)

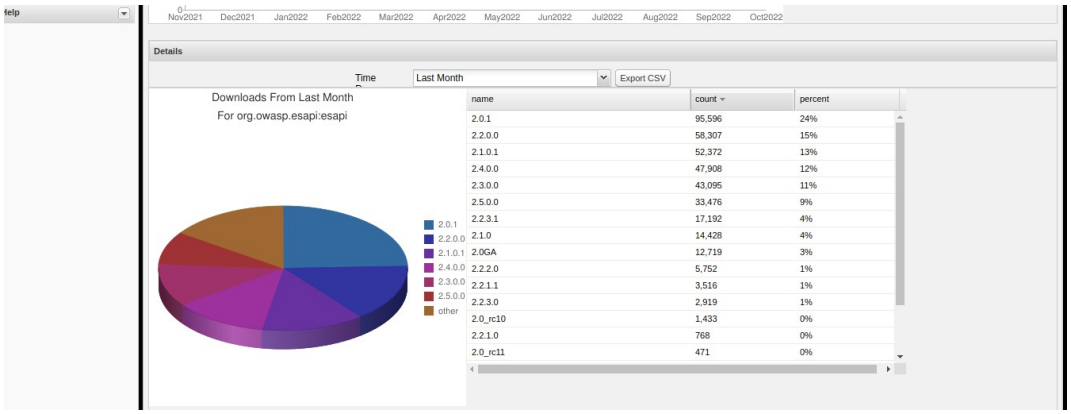


This is a screenshot from Maven Central made on 11/27/2022, the date I released ESAPI version 2.5.1.0. It shows the historical downloads of all the ESAPI versions for a 12 month period, from Nov 2021 through Oct 2022. Note at this point, the previous version, 2.5.0.0, was just over 4 months old.

The next slide is the bottom part of this same screenshot and it shows a breakdown by versions of (new) downloads), but only for the previous month (Oct 2022).

We could fix this problem if we were just allowed to delete old versions from these code repositories, but Maven Central does not allow that, so there is no way to prevent Maven, Gradle, Ivy and other tools that work with Central to prevent them using old pinned versions. Maybe those tools could scream warnings for direct dependencies at least that you are using an old version the **first** time you download it, but that is not the default behavior. They "***all download artifacts from a release repository once. They don't come back to a release repository and check for a newer version***." All later retrievals are from your local cache. [Reference 2]

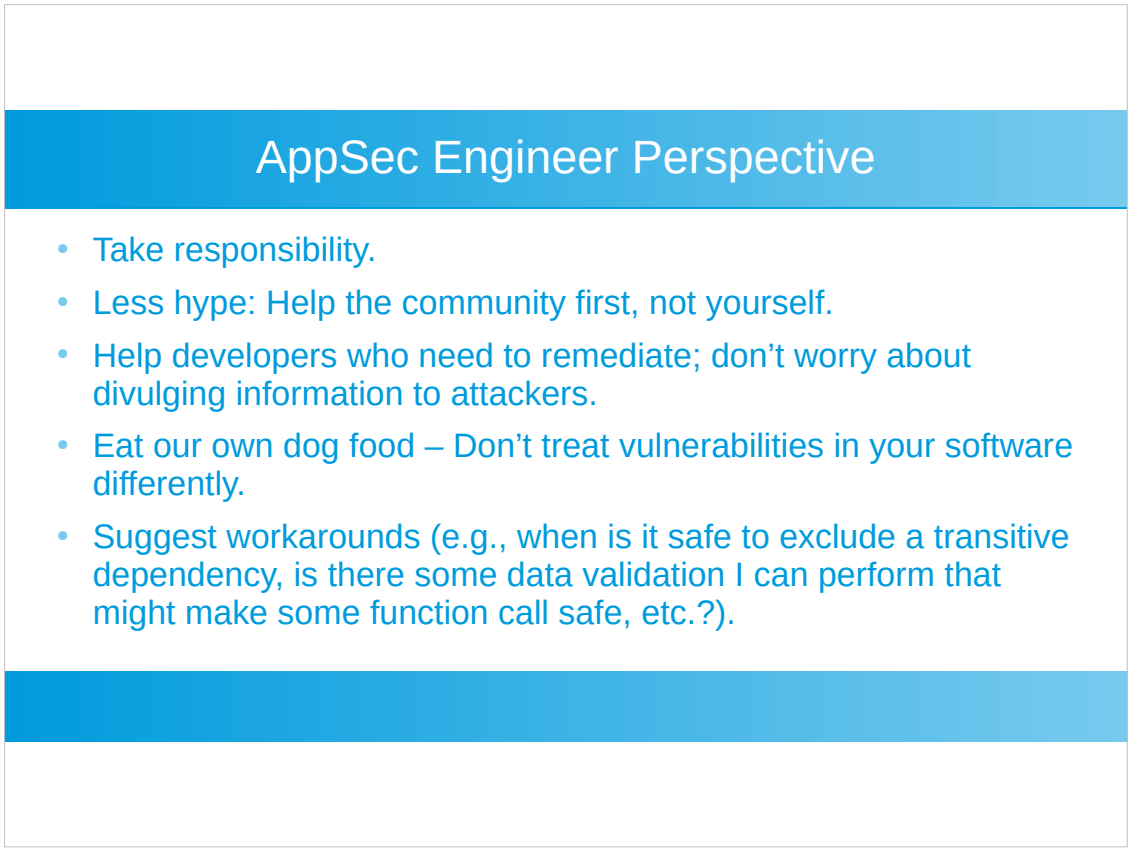
Evidence That Many Are Not Patching (2/2)



For Oct 2022: This is a snapshot just after 2.5.1.0 was released, but 2.5.0.0 was released 4 full months earlier and it only amounts to 9% of the downloads. But contrast, the 2.0.1 version was released in 7/2011 and accounts for 24% of the downloads and the 2.2.0.0 release was released in 6/2019 and was 15%!

Moral: If you use another library, then try to use the most recent library.

- If you develop a library that others use, do NOT just abandon it.
- Try to find someone else to take it over.
- Try to set up with Dependabot or Snyk, etc. to create PRs and get someone to approve them and merge them and periodically do a release.
- If you can no longer support a project, then ARCHIVE the damned project so that others know that it is abandoned and new projects don’t start using it.

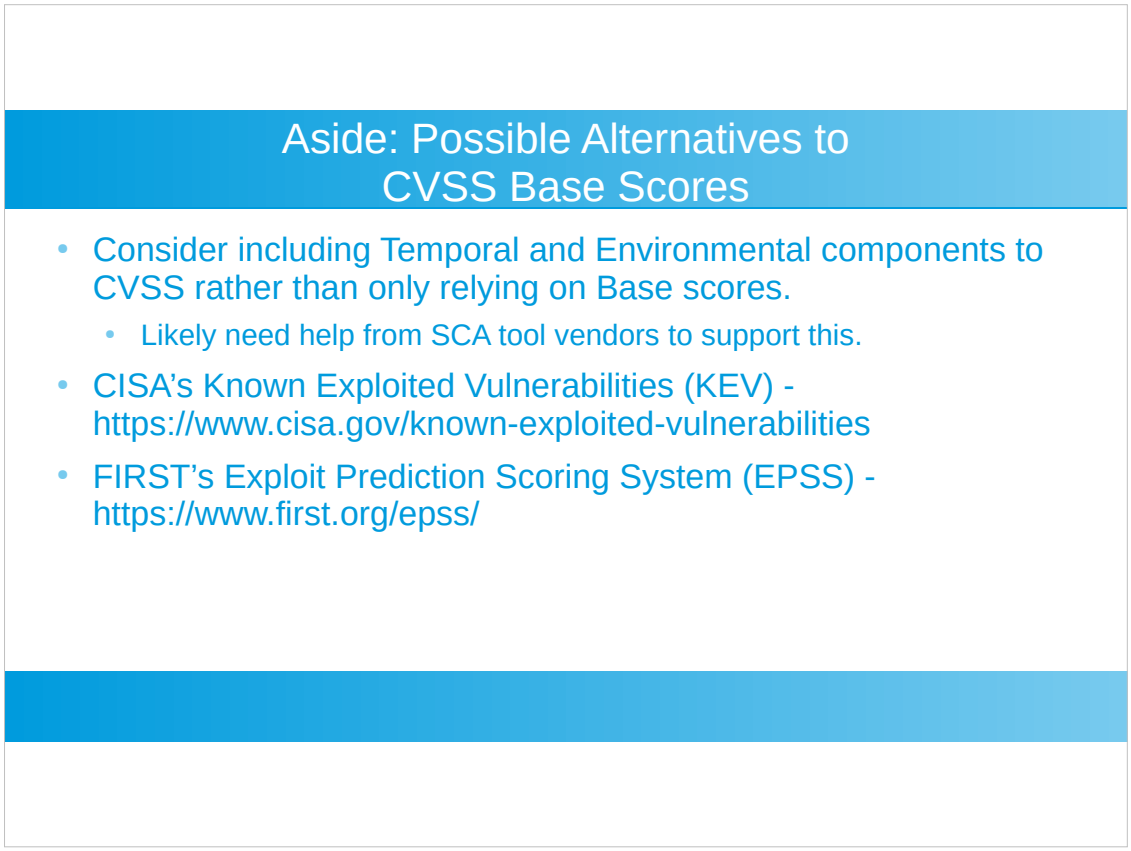


There are too many of AppSec wannabes who want to get a name for themselves. They discover a vulnerability, and then provide little to no details on how to reproduce the problem. They cause "noise" in the system.

I realize that many pen testers don't know how to code in some particular language X, but if you don't, help the dev team to write up a GitHub issue, possibly a CVE description, provide workarounds, and ideally, create a PR for them to remediate the issue.

Realize doing the right thing is mostly a thankless job. (If you know that, you won't be disappointed when you don't get it.)

Maybe we need a "helpfulness" reputation system for AppSec people. IDK.



- CISA is the US government's Cybersecurity & Infrastructure Security Agency.
  - Designed to help prioritize remediation efforts.
  - Criteria: An assigned CVE ID & active exploitation.
  - Active exploitation: "reliable evidence that execution of malicious code was performed by an actor on a system without permission of the system owner".
  - 981 entries as of August 6, 2023, most against standalone executables software; a minority are libraries.
- FIRST is Forum of Incident Response and Security Teams. It is the organization that came up with CVSS.
- Even if SCA tools didn't want to abandon use of CVSS entirely, they could augment their threat priorities with one of these. Just a thought.

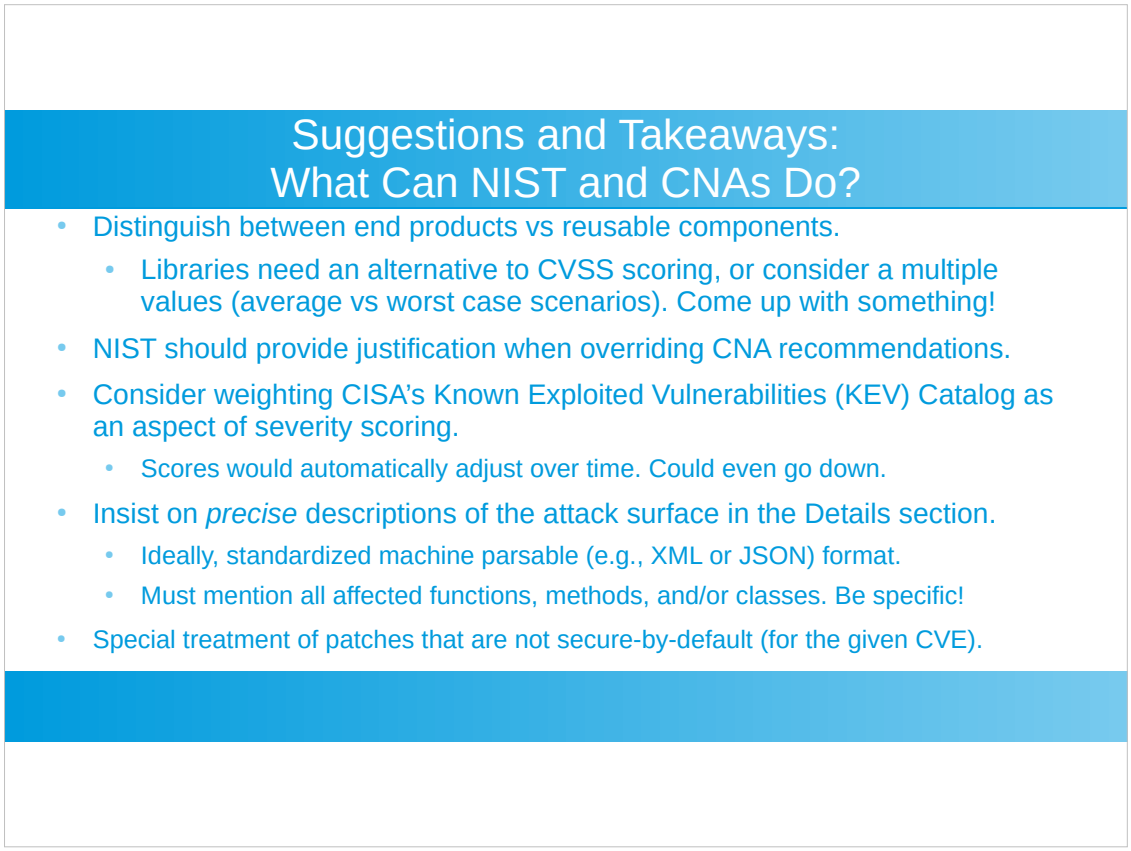
I realize that the problem with some of these suggestions is that they might require intensive manual effort unless your SCA tool vendor steps up and adds some type of "reachability analysis"; however, that needs to be weighed against the effort you are forcing on all your development teams to needlessly patch and re-run their regression tests and hope that patching some transitive dependency is not going to break something. (In other words, stop locally optimizing what is best for your production engineering, security, and operations team and consider the impact vs effort from a more holistic perspective.)

Don’t worry OWASP, I’m not going to call out any specific vendors. After all, some of them may be our sponsors. :)

If you do want to include findings from your private vulnerability databases, include things like commit IDs or PRs or git diffs where these claims can be substantiated and stop hyping the severity of the risk. The who CVSS system already does that, so please don’t make it worse.

If your SCA product doesn’t already include “reachability analysis” (which requires you to look at source code *usage*, not just dependency trees), than seriously consider adding it. Even better, develop *runtime* SCA monitoring!

Consider issuing warnings for patched libraries that are not secure-by-default. (Admittedly, this is tough w/out NVD’s help.)



Last bullet:

- When the patch is not secure-by-default, then If the actual vulnerability is in (say) some super class or an abstract class, then all CTORs and/or inherited methods for all subclasses should be required to be mentioned as well.
- If the patch is secure-by-default, while that is a "nice to have", it is not as critical.

RECOMMENDATION: If a patch to a CVE is not secure-by-default, consider requiring a separate, prominent section (that can be checked via your API calls) where the additional instructions must be placed to describe what must be done to make it secure.

References

- Jeff Williams; “What about Transitive Dependencies?” LinkedIn post (2023-07-27) - https://www.linkedin.com/posts/planetlevel_what-about-transitive-dependencies-i-hear-activity-7090447433146515456-n4GW
- Question “Can I change a component on Central?”, https://web.archive.org/web/20201112013316/https://central.sonatype.org/articles/2014/Feb/06/can-i-change-a-component-on-central/ (posted 2014-02-06, retrieved 2023-09-02).
- GitHub Security Advisory: DoS vulnerabilities persist in ESAPI file uploads despite remediation of CVE-2023-24998, https://github.com/ESAPI/esapi-java-legacy/security/advisories/GHSA-7c2q-5qmr-v76q

This only includes references that I didn’t already provide links for in the main part of the talk.

Questions?

- Ask me now, or drop me an email at kevin.w.wall@gmail.com
- Or, if you don't mind waiting a bit, DM me as @KevinWWall on OWASP Slack or Twitter (although the latter may take a while).

NOTE: Slide deck, with speaker's notes, available at:

https://github.com/kwwall/presentations/tree/master/SBOMs_to_F-Bombs

I rarely go to Twitter any more. Maybe only once a month, So use that only if you don't mind waiting for an answer.

So, questions?


OWASP 2023
GLOBAL
AppSec
WASHINGTON
DC
OCT 30 • NOV 3
THANK YOU