RSA Conference2016
San Francisco | February 29 – March 4 | Moscone Center

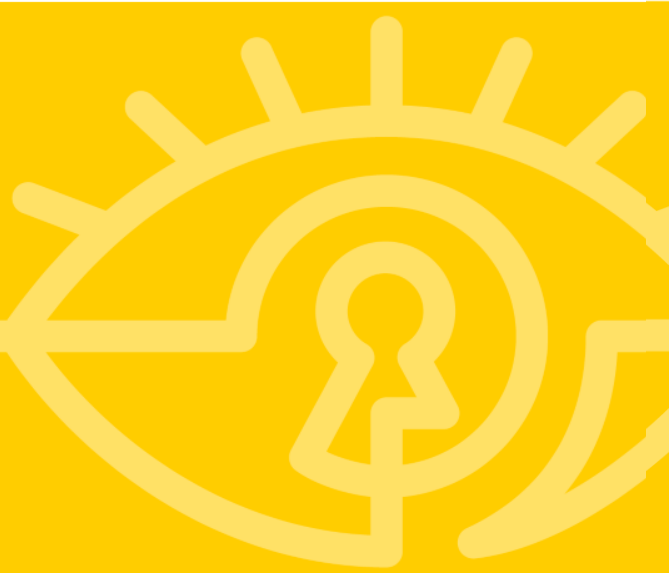
Connect to Protect

SESSION ID: ASD-F02

Open-Source Security Management and Vulnerability Impact Assessment

Gunter Bitz
Senior Manager Legal Compliance
SAP SE

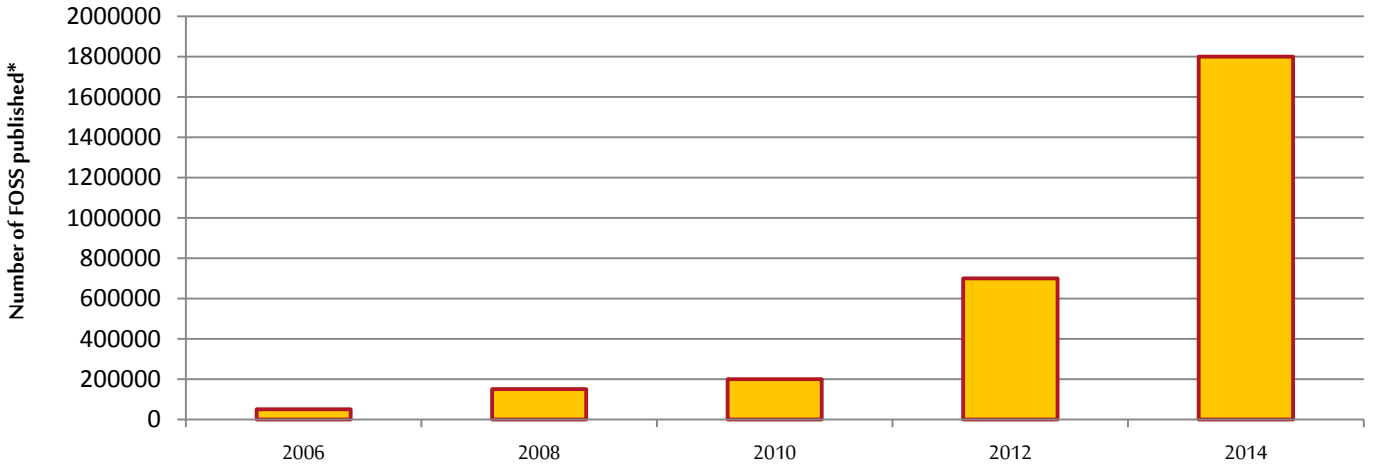
Henrik Plate
Security Architect
SAP SE



Open-Source Security Management Introduction

Published Free and Open Source Software (FOSS)

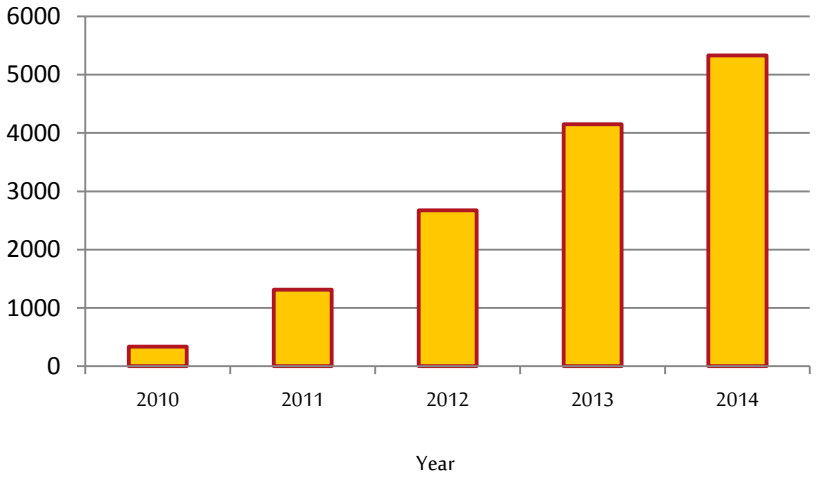
- 1.5 fold increase in FOSS every year, currently 2M [Black Duck 2014]
- 1.2 billion by 2030 [Internet of Things (IoT) and Smart Planet (IBM)]

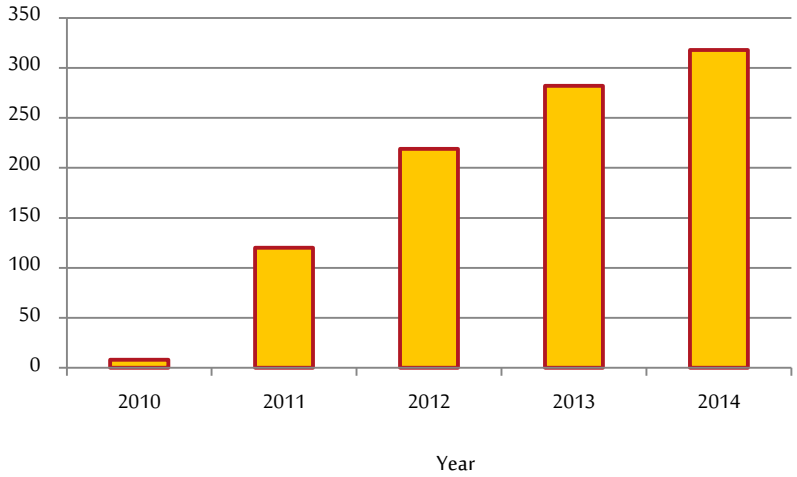


*Based on a report by Black Duck Software

Increase of FOSS usage for SAP product development

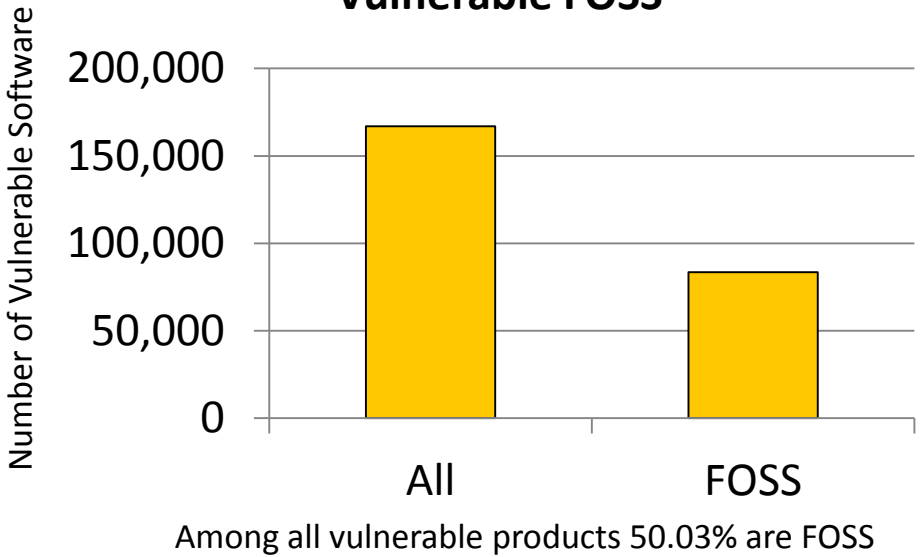
- Number of FOSS versions (used at SAP) increased by 10 fold
- Number of SAP programs (that use some FOSS) increased by 40 fold





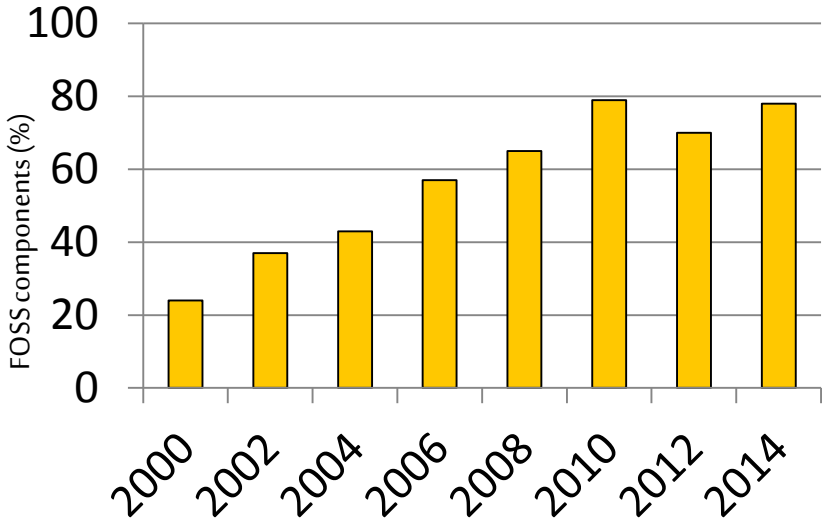
Share of FOSS Components with vulnerabilities

All Vulnerable Products vs. Vulnerable FOSS

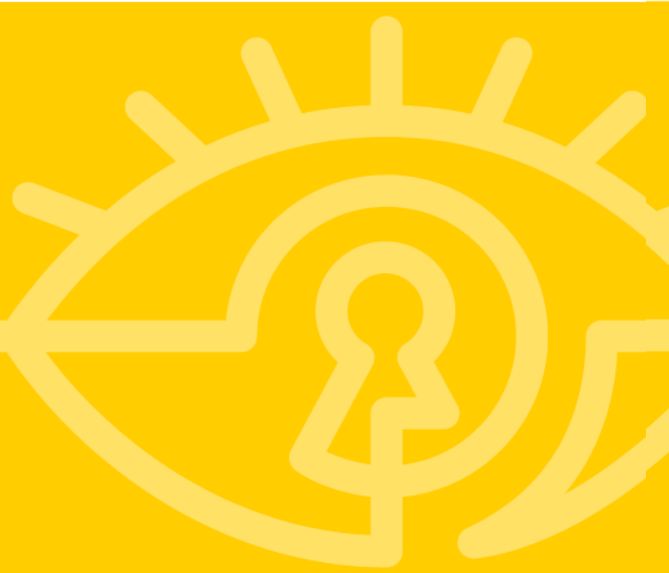


Among all vulnerable products 50.03% are FOSS

Percentage of vulnerable FOSS software is increasing



Based on the National Vulnerability Database (1999-2014) published by the National Institute of Standards and Technology (a repository by the U.S. government)



Open-Source Security Management Project Report

Project report how to manage vulnerable FOSS at SAP

- Project at SAP conducted 2014 – 2015
- 2 sub projects:
  - Define a process to fix vulnerable FOSS components in new products / new release before they are made available to customers
  - **Address vulnerabilities in software products already shipped (made available) to customers**
- Driven by central Legal Compliance & Product Security teams

Organizational Set-up

Board Member for Products and Innovation

Escalation Level

Area Head | Area Head | Area Head

Head of Architecture, Security and Compliance

Steering Level

Product Teams

Product Security

Legal Compliance

Working Level

Note: Complexity is often added through M&A activities. Might be difficult to „reach“ new teams. Some are not using same standards and policies yet….

A recipe for managing FOSS (in)security

- Ingredients:
  - List of all FOSS components used in all software products
    - At SAP: Workflow based FOSS request process.
    - All approved FOSS components stored in a database (can be retrieved per product)
  - List of vulnerabilities in FOSS components – e.g. „NVD“ from NIST* (XML data feed available and used for automation)
- Tools:
  - A database (any kind)

SAP

* https://nvd.nist.gov/

A recipe for managing FOSS (in)security

- Cooking instructions
  - Map internal FOSS name to CPE identifier used in NVD (or wait for Black Duck and NIST to do that for you*)
    - Example: **org.apache.commons:commons-compress:1.4** (Maven GAV) needs to match to **cpe:/a:apache:commons-compress:1.4**
  - Use CPE name to lookup vulnerabilities for each FOSS component in your product(s)
    - Challenges: Multiple usage of same FOSS component (or different version) in the product
  - Identify person responsible for software product (internal product database)
  - Notify person responsible and provide list of vulnerabilities for the product
  - Ask person responsible to analyze the exploitability of the vulnerabilities
  - Provide a fix as necessary
    - Patching is fundamentally different for Cloud Software (vs. „on premise“ products)

* We have suggested that NIST adds CPE identifiers for FOSS even there hasn‘t a vulnerability been reported yet. Black Duck inc. is providing FOSS data from their knowledge base (work in progress).

Typical challenges for product owners

- No one in the team is familiar with the inner workings of the (vulnerable) FOSS component
  - Business case for usage of FOSS often does not account for additional maintenance costs of code no-one is familiar with.
- Difficult to assess exploitability in the actual context the FOSS is used in the product.
  - 95.7% of OSS with vulnerabilities have a newer version which fixes the problem*
  - BUT Number of individual patches should be minimized as the efforts on customer's side multiply.
  - Fear of updating FOSS components to a later, non vulnerable version due to potential incompatibilities.
- Other priorities (e.g. new feature requests)

* http://www.whitesourcesoftware.com/infographics-open-source-great-use-right/

Vulnerability assessment: Help needed!

- Is the bill of material (in regards to FOSS usage) correct? Is the version number correct?
- Is the vulnerable code actually present in the product provided to the customer (Example: Only JavaScript engine used but the Firefox package was requested).
- Is the vulnerable function ever called? If you say „no“ can you make sure that there is no way to call it by manipulating data from outside?
- **Academic question:** If a vulnerable function is called: Which parameters are needed to exploit the vulnerability? Can an attacker influence the software in this way?
  - HIGHLY RISKY in case you say „no“ but do not patch or upgrade

SAP

Green bullet points are now automated at SAP (shown in next section)

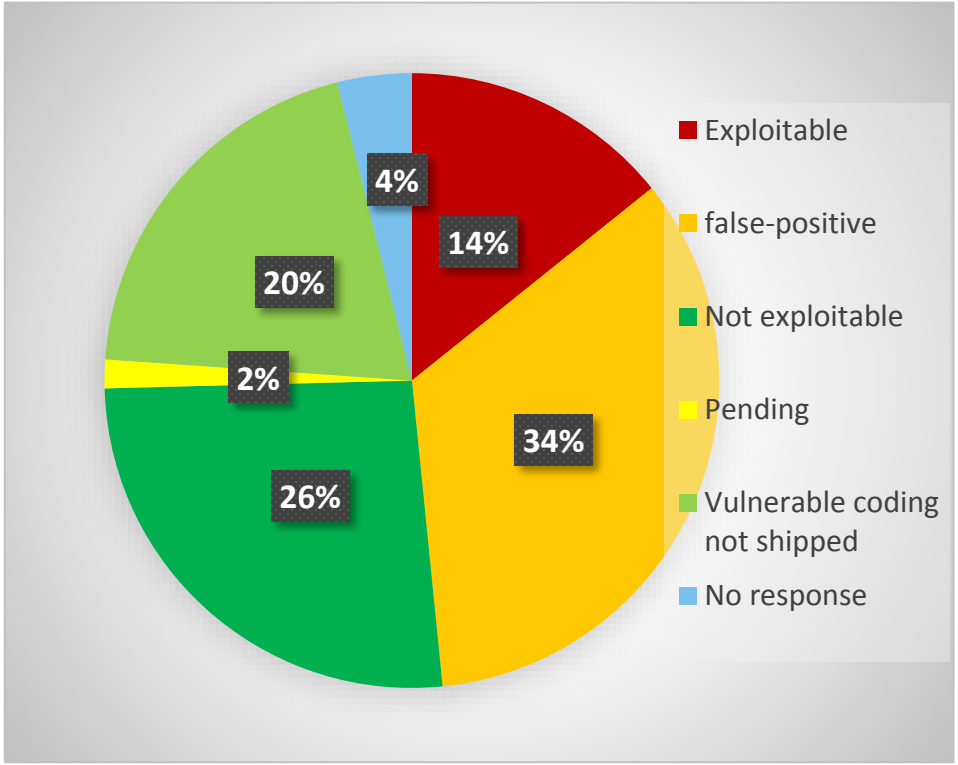
What else helped?

- Regular reporting to management team on project progress
- Follow-up on the „no response“ cases. We needed up to 6 reminders
- Escalation to management for „no response“ cases

Long term:

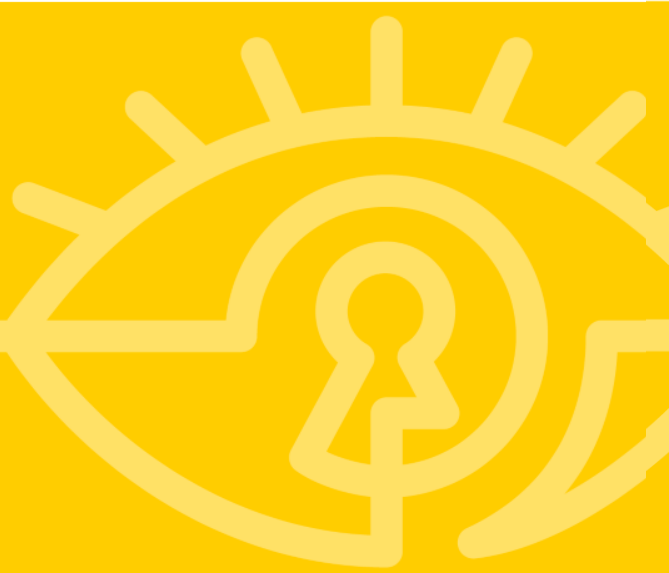
- Adding the topic of vulnerable FOSS to the development standards and guidelines and to release decision checklist
- Using functionality provided by Open Source Management tools

A Snapshot after 7 months



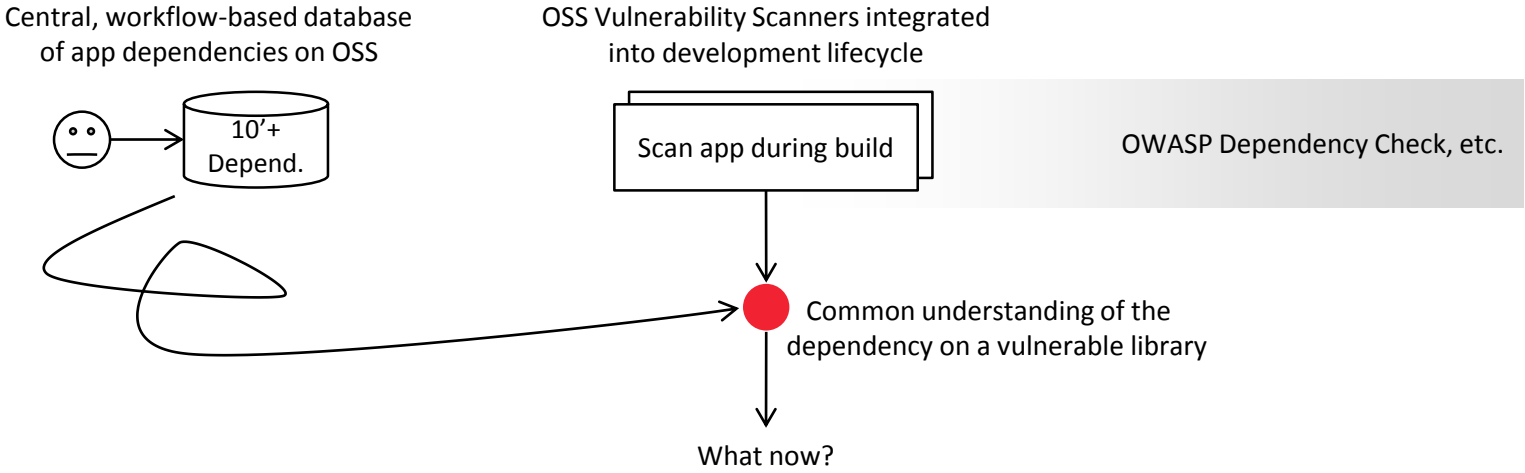
- Exploitable: Confirmed that the vulnerability in the FOSS component can be exploited.
- False-Positive: Due to wrong meta-data. E.g. other FOSS version used than specified
- Not exploitable: Confirmed that the vulnerability in the FOSS component can NOT be exploited. BUT the vulnerable code exists.
- Pending: Product team has not yet finished the analysis
- Vulnerable coding not shipped: The code containing the vulnerability of the FOSS is not present in the product
- No response: Product team did not respond to e-mails and escalations

Snapshot taken from a project in progress 7 months after kick-off

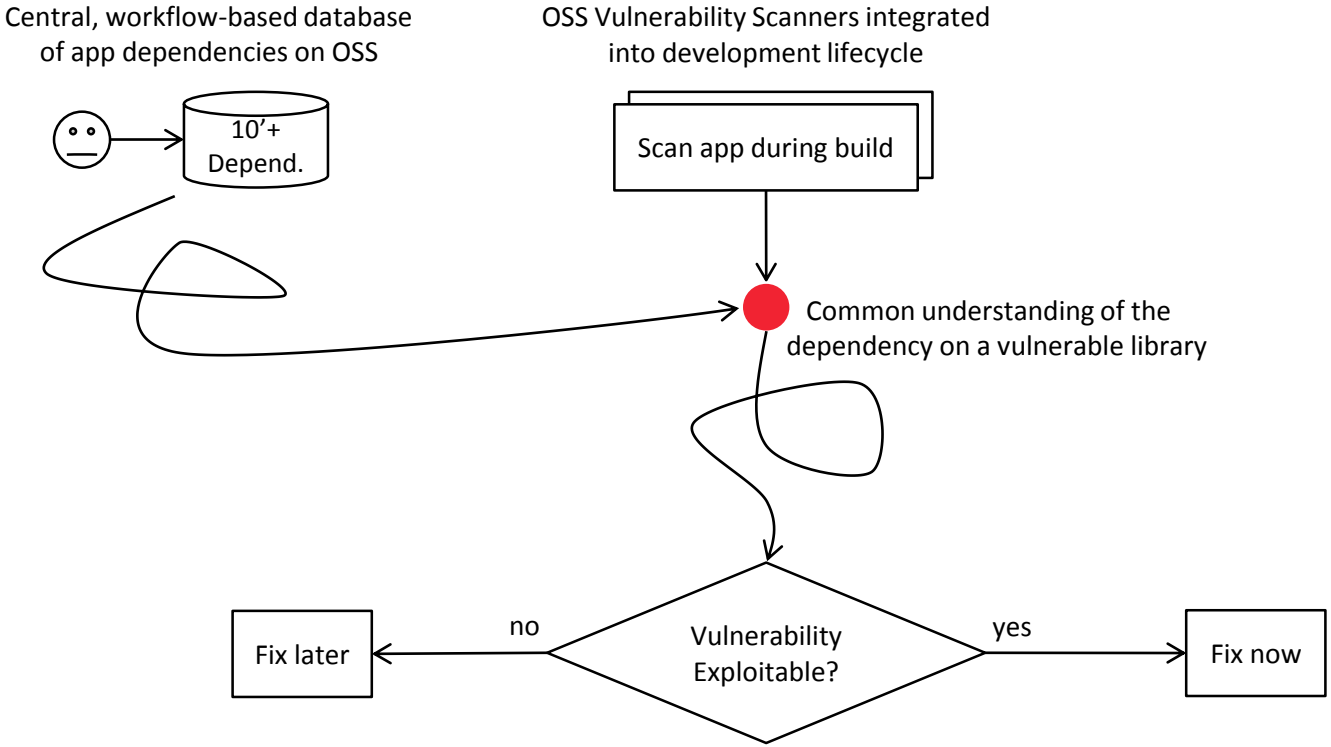


Vulnerability Impact Assessment

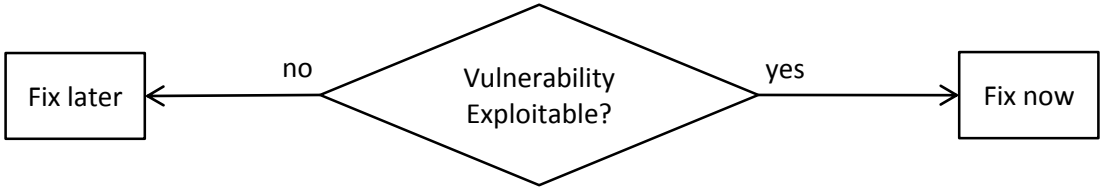
You Include a Vulnerable Library – What Now?



Solution Goal – Assess Exploitability

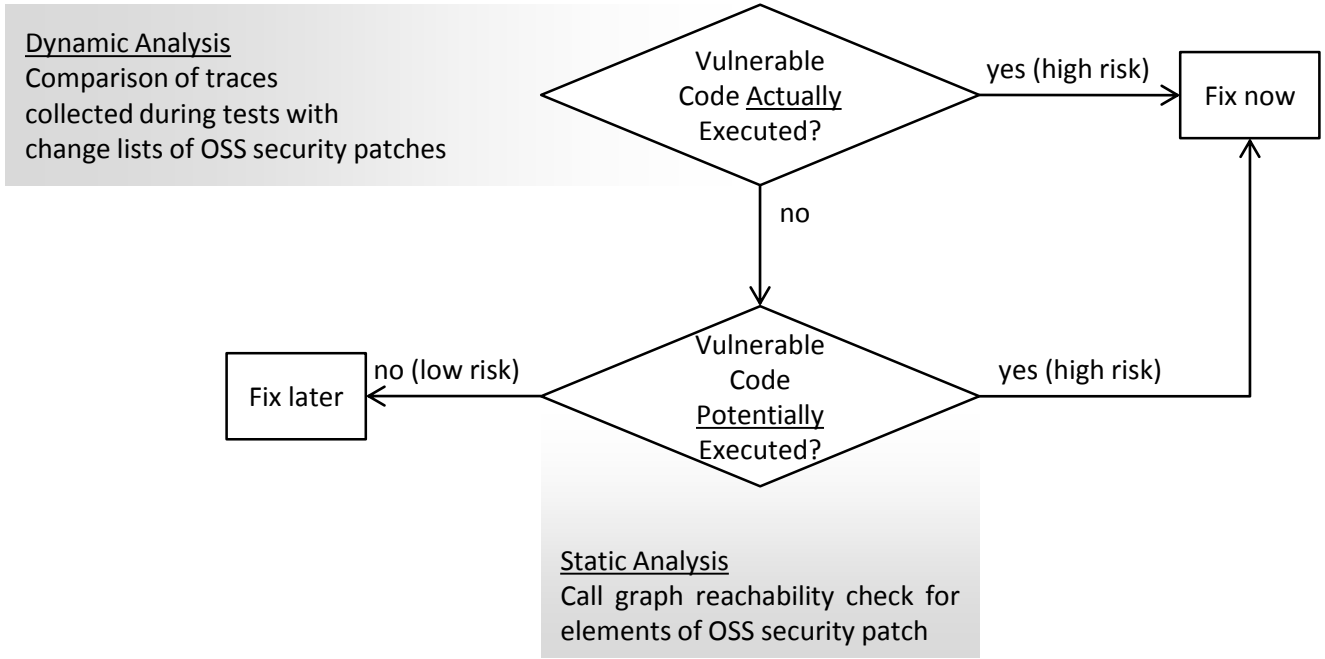


Solution Approach



- Application-specific exploitability is difficult to determine (minimalistic vuln. descriptions, transitive dependencies, multi-module OSS projects, data provenance, sanitizations, configurations, etc.)
- Only code matters: Can the application be executed in such a way that vulnerable library code is ran?
- Assumption: If an application executes code for which a security fix exists, then there is a significant risk that the vulnerability can be exploited in the specific application context

Solution Approach



Plate, Ponta, Sabetta, "Impact assessment for vulnerabilities in open-source software libraries," ICSME 2015, 31st IEEE International Conference on Software Maintenance and Evaluation

Assessment Levels

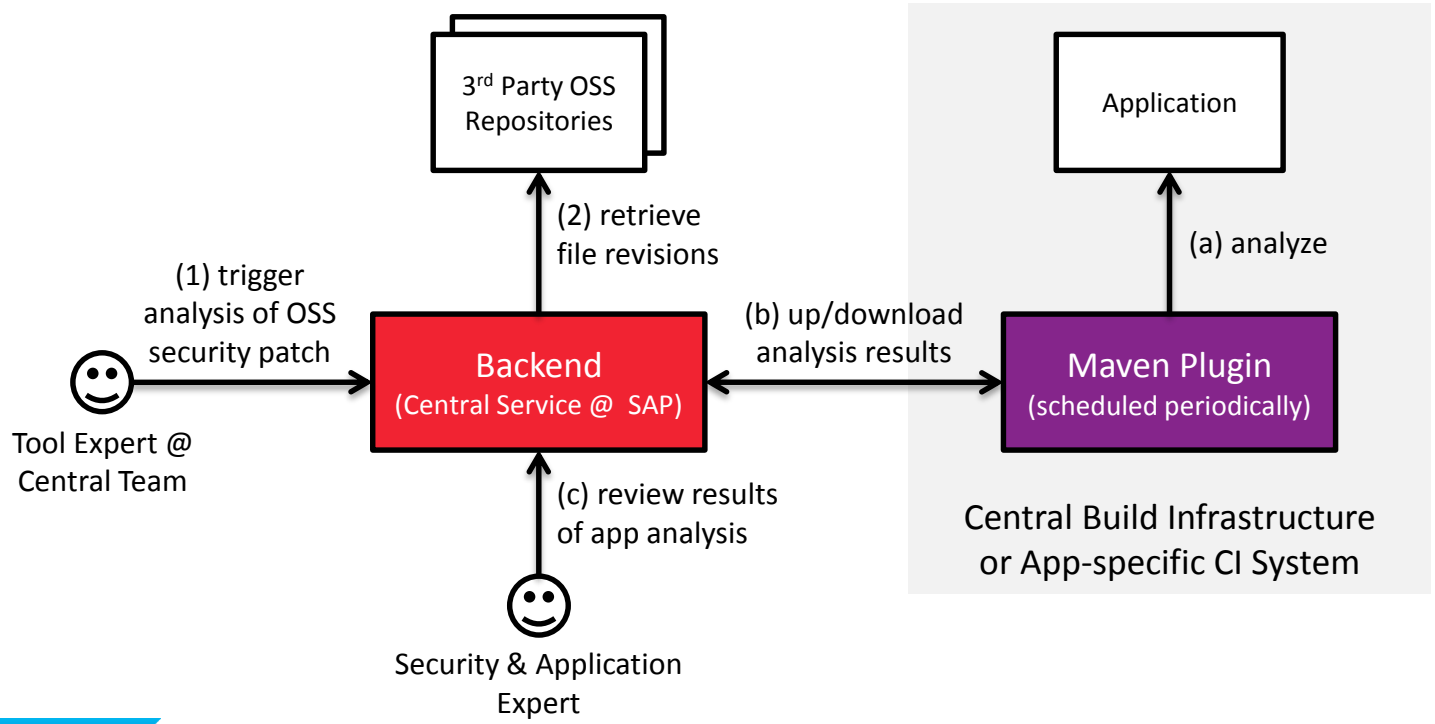
Non-vulnerable library release used

Vulnerable library release used [1]

Vulnerable library code potentially executable [2]

Vulnerable library code actually executed

Solution Architecture (Java)



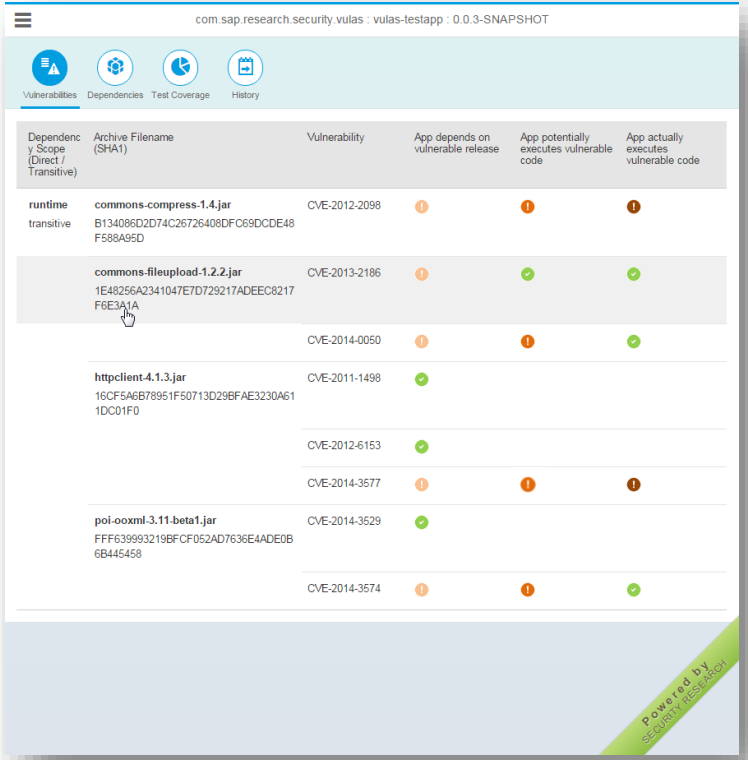
Example & Screenshots

CVE-2012-2098

- Algorithmic complexity vulnerability in the sorting algorithms […] (BZip2CompressorOutputStream) […] allows remote attackers to cause a DoS […].
- cpe:/a:apache:commons-compress:*

Maven GAV

- org.apache.commons : commons-compress : 1.4



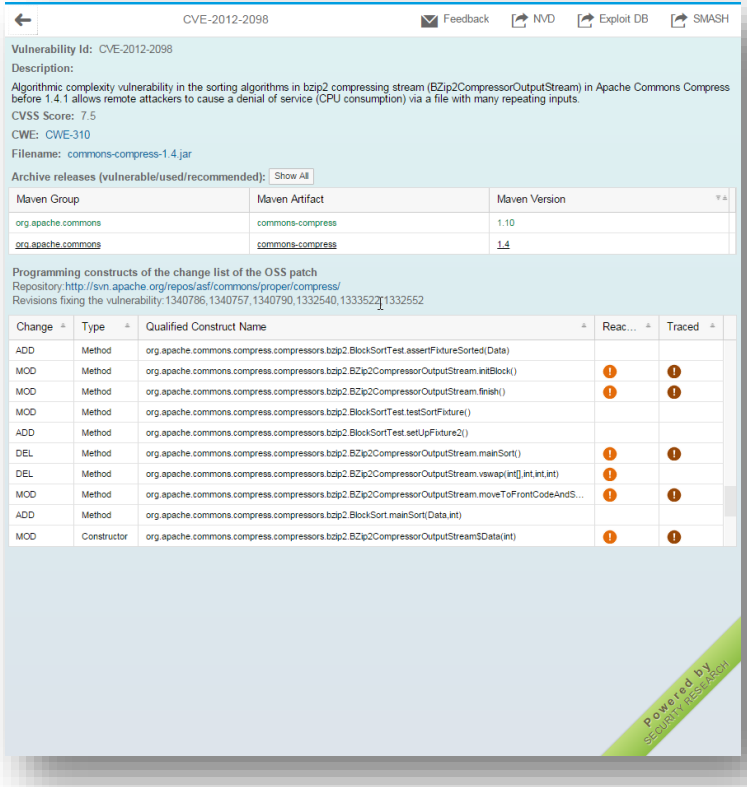
Example & Screenshots

CVE-2012-2098

- Algorithmic complexity vulnerability in the sorting algorithms […] (BZip2CompressorOutputStream) […] allows remote attackers to cause a DoS […].
- cpe:/a:apache:commons-compress:*

Maven GAV

- org.apache.commons : commons-compress : 1.4



Example & Screenshots

CVE-2012-2098

- Algorithmic complexity vulnerability in the sorting algorithms […] (BZip2CompressorOutputStream) […] allows remote attackers to cause a DoS […].
- cpe:/a:apache:commons-compress:*

Maven GAV

- org.apache.commons : commons-compress : 1.4

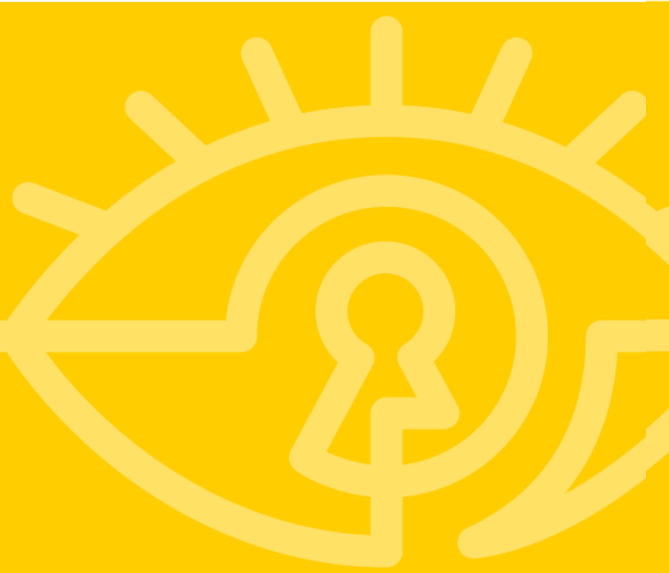
Wrap-up & Outlook

Today

- Code-centricity reduces false-positives, and is robust against rebundling
- Static and dynamic analyses prioritize backlog
- New bugs do not require new scans
- Productively used at SAP

Tomorrow

- Continued innovation, e.g., as part of EIT project VAMOSS
- Production of re-usable library call graphs
- Introduction of risk metrics
- Analysis of alternative fixing strategies



Summary & Apply

Apply What You Have Learned Today

ANALYZE

- Keep track of your applications' BoM and map their items to publicly known vulnerabilities
  - Preferred: Use tools integrated into the build process
  - WARNING: For large projects you might find thousands but it will help you to get management attention!

MANAGE

- Define a decision-making process for the production of application patches (Q: Now or later?)
  - Criteria: Deployment models, shipment status, exploitability, etc.

DO

- Use dynamic and static analysis to assess the exploitability of vulnerabilities

Contact Information

Dr. Gunter Bitz, CISSP, CPSSE

gunter.bitz@sap.com

+49 6227-768765

Henrik Plate, CISSP

henrik.plate@sap.com

+33 4 9228-6348