FOSSA + OWASP LA

July 24th, 2024

Speaker: Chelsea Boling

FOSSA

HOWDY! Who likes an agenda?! 🚀🚀🚀🚀🚀

- ❑ Intros
- ❑ CVE noise and breaking it through
- ❑ Vulnerability prioritization demonstration

Breaking Through CVE Noise: Analyzing 5 Key Prioritization Inputs

Streamlining Vulnerability Resolution Process

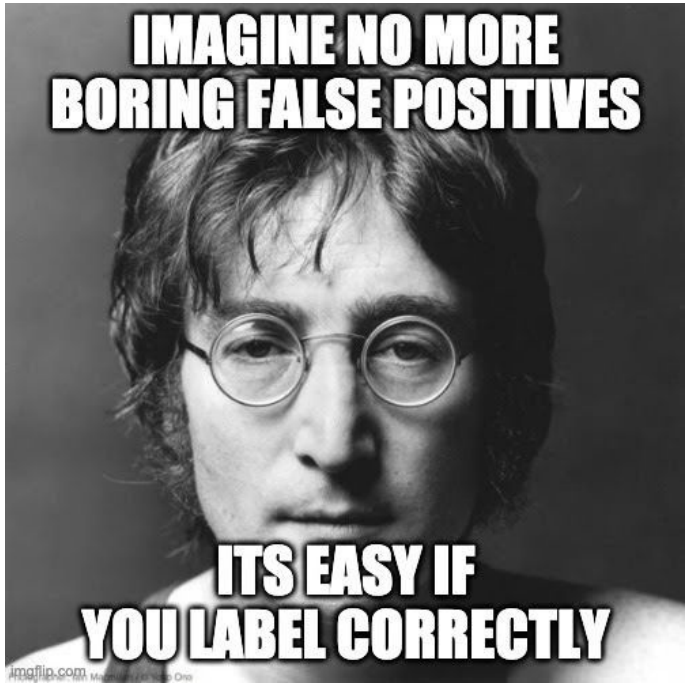
FOSSA

Understanding CVE Noise

Managing thousands of CVEs poses significant challenges for security teams in prioritizing threats effectively.

With thousands of new CVEs reported annually, security teams face the daunting task of sifting through and prioritizing the most critical vulnerabilities.

Distinguishing between the severity levels of vulnerabilities is crucial to allocating resources efficiently and addressing the most impactful threats first.



CVSS Scores

The Common Vulnerability Scoring System (CVSS) provides a standardized method for assessing and prioritizing vulnerabilities based on their severity and impact on systems: https://nvd.nist.gov/vuln-metrics/cvss

EPSS Scores

The Exploit Prediction Scoring System (EPSS) helps security teams predict which vulnerabilities are likely to be exploited and need immediate attention, enabling proactive security measures.

CISA KEV Catalog

The CISA (Cybersecurity & Infrastructure Security Agency) Known Exploited Vulnerabilities (KEV) Catalog helps identify vulnerabilities actively exploited in the wild, guiding security teams in focusing on immediate threats with known exploit activity.

VEX Data

Vulnerabilities Exposure Factor (VEX) communicates whether or not a vulnerability is exploitable given it's real world use case and its potential impact across the business organization.

Reachability Analysis

Reachability analysis infers if the vulnerability is reachable in your first-party code. Reachability analysis lists call paths from your first-party code to vulnerable functions associated with CVEs. This way, you can proactively remediate the issue by modifying your usage of dependency. This highly depends on your build environment.

Business Context

Prioritization of vulnerabilities comes with what you may know in your own business (e.g. the types of security tools that are available (or not available), the cybersecurity budget your organization you may have, the applications that are the most valuable to your org).

CVSS Scores

*CVSS (Common Vulnerability Scoring System) has long been seen as the de facto mechanism for vulnerability prioritization, but industry focus in recent years has shifted to **exploitation** as an indicator for vulnerability risk*

GitHub Advisory Database / Unreviewed / CVE-2023-35116

An issue was discovered jackson-databind thru 2.15.2...

Unreviewed | Published on Jun 14, 2023 to the GitHub Advisory Database • Updated on Nov 4, 2023

Package

No package listed— Sugge

Description

An issue was discovered ja
crafted object that uses cy

References

- https://nvd.nist.gov/vul
- FasterXML/jackson-dat

Published by the Nationa

Published to the GitHub

Last updated on Nov 4, 2

Severity

High 7.5 / 10

CVSS base metrics	
Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	None
Integrity	None
Availability	High

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H

Projects such as EPSS and CISA Known Exploited Vulnerabilities (KEV) have emerged, providing visibility into this data for the entire security community.

EPSS Scores

CRIT 10 | 97.6% (100th) | No fix | CVE-2021-44228 | Exploit known

Improper Input Validation

in log4j:log4j (1.2.14)

Active in 5 projects

Ignore | Create ticket

Issue | Projects 5 | Comments 0

Remediation

Current version 1.2.14 — CURRENT VERSION

You are exposed to this vulnerability with this version.

No fix available

FOSSA cannot find a fix for this vulnerability at this time.

Vulnerability details

CVE	CVE-2021-44228
CWEs	CWE-20 CWE-400 CWE-502
EPSS Score	97.6% (100th)
Affected versions	<2.15.0
Patched versions	2.12.2 2.12.3 2.3.1
Publication date	Dec 10, 2021
Review status	Reviewed

Example of KEV Data

https://www.cisa.gov/known-exploited-vulnerabilities-catalog

VMWARE | VCENTER SERVER

CVE-2022-22948

VMware vCenter Server Incorrect Default File Permissions Vulnerability : *VMware vCenter Server contains an incorrect default file permissions vulnerability that allows a remote, privileged attacker to gain access to sensitive information.*

Known To Be Used in Ransomware Campaigns? **Unknown**

Action: Apply mitigations per vendor instructions or discontinue use of the product if mitigations are unavailable.

- **Date Added:** 2024-07-17
- **Due Date:** 2024-08-07

Additional Notes +

Vulnerability Exploitability Exchange (VEX)

VEX (Vulnerability Exploitability eXchange) is a set of formats used to describe whether vulnerabilities that affect components of a software product affect the product itself.

Why is VEX important?

The vast majority of vulnerabilities actually aren't exploitable in their real-world product context.

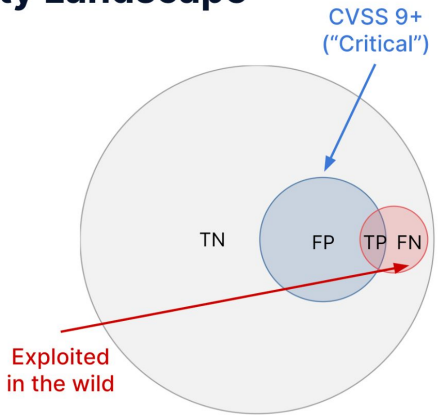
Reasons why include that:

- The vulnerable component isn't present
- The vulnerable code isn't present
- The vulnerable code can't be controlled by an adversary
- The vulnerable code isn't in the execute path
- Inline mitigations already exist

The Vulnerability Exploitability Landscape

Effort: What percent of published vulnerabilities were prioritized?

effort = (FP + TP) / (TN + FP + TP + FN)



References
TN = True Negative | FP = False Positive | TP = True Positive | FN = False Negative

All Vulnerabilities

VEX data (which is provided by the software supplier) complements other vulnerability inputs (like EPSS, CVSS, and reachability analysis) to provide a more accurate picture of security risk.

https://www.cisa.gov/sites/default/files/publications/VEX_Status_Justification_Jun22.pdf

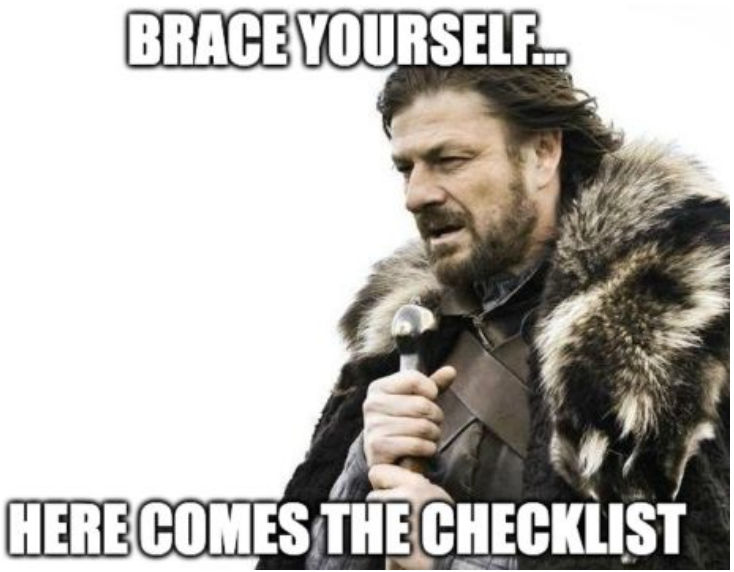
Vulnerability Exploitability Exchange

VEX Data


```
"published": "2024-03-06T22:15:57.000Z",
"affects": [
  {
    "ref": "pkg:deb/ubuntu/libgcrypt20@0:1.10.2-3ubuntu1?arch=amd64&distro=ubuntu-23.10",
    "versions": [
      {
        "version": "amd64#0:1.10.2-3ubuntu1",
        "status": "affected"
      }
    ]
  }
],
"analysis": {
  "state": "not_affected",
  "justification": "Inline mitigations already exist",
  "detail": "Inline mitigations exists outside the usage of this package ensuring our production deployment is not
  affected by this vulnerability. Please review - https://dave-fossa.atlassian.net/browse/DAVE-172, for detailed
  analysis. "
}
```

Answer the basic, adminy questions first

- ❑ Do I know the high level functions of the application(s) that I support? List these functions.
- ❑ What programming languages, frameworks and package managers are used?
- ❑ How well do I know about the open source packages that are used in the applications?
- ❑ Are there existing processes in place to block code changes if there are vulnerabilities present?



Recommendations

- ❑ Look at vulnerabilities identified in **direct dependencies**
- ❑ Filter by **CVSS score**
- ❑ Is there a fix available?
  - ❑ Start with *patches* or *minor* upgrades
- ❑ Focus on vulnerabilities listed in the **KEV catalog**
- ❑ Sort by the **highest EPSS score** within the *detected KEV CVEs*.

Next level recommendation

Explore/review results from **reachability analysis** (for contextual risk assessment)

- Previous inputs may or may not apply

CRIT 9.8 | 0.7% (80th) | Reachable | Patch fix | CVE-2020-10683

Improper Restriction of XML External Entity Reference

in org.dom4j:dom4j (2.1.0)

Active found 4 months ago

Ignore | Create ticket | View Path

Issue | Projects 3 | Reachability | Comments 0

Reachable functions

The vulnerability is reachable via the functions identified below.

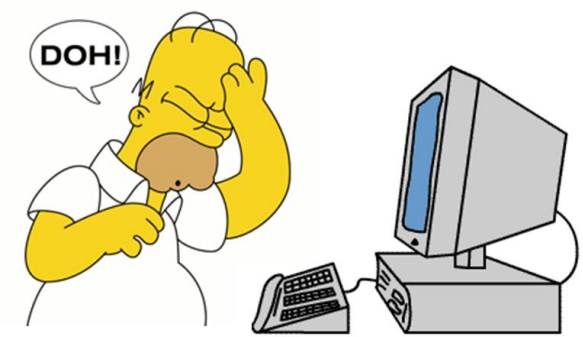
1 function identified

```
1. org.dom4j.io.SAXReader:<init>()

/Users/halmeoni/Desktop/reachability-with-maven-example/target/example-1.1.jar (:)
└ com.example.app.App:main(java.lang.String[])
  └ com.example.app.utils.SomeOtherReader:parse(java.net.URL)
    └ org.dom4j.jaxb.JAXBReader:read(java.net.URL)
      └ org.dom4j.jaxb.JAXBReader:getReader()
        └ org.dom4j.io.SAXReader:<init>()

/Users/halmeoni/Desktop/reachability-with-maven-example/target/example-1.1.jar (:)
└ com.example.app.App:main(java.lang.String[])
  └ com.example.app.App:parse(java.net.URL)
    └ org.dom4j.io.SAXReader:<init>()
```

Reachability analysis is not perfect. The identified functions are statically discovered. There may be additional dynamic instances of vulnerable functions that FOSSA can't find (eg. after the project is built). The context of how this function is used may also be important to determining if this finding is valid as reachable.

More at Docs/Reachability

Let’s tackle vuln overload in an actual SCA tool.....



Oh, yeah, like FOSSA

Some takeaways

Combine Severity and Exploitability to Assess Risk

Use **CVSS scores** for severity and **EPSS scores** for the likelihood of exploitation to assess overall risk.

Prioritize Actively Exploited Vulnerabilities

Prioritize vulnerabilities in the **CISA KEV catalog** and use **VEX data** to understand exploitability and mitigation availability.

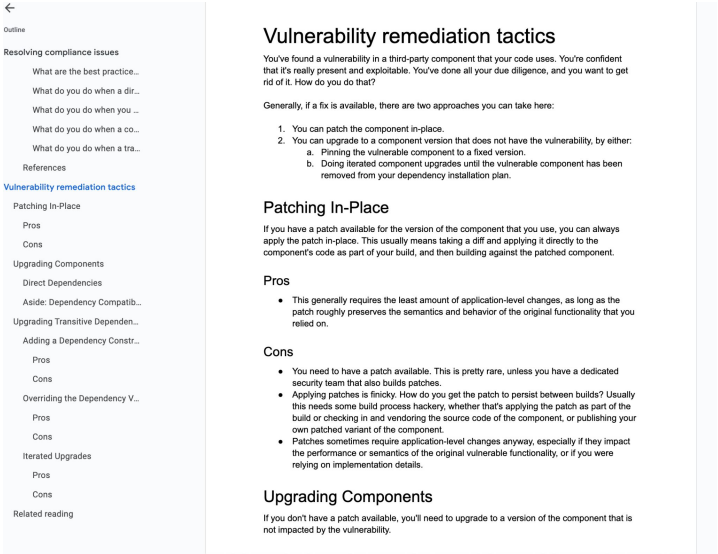
Contextualize with Reachability Analysis

Incorporate **reachability analysis** to determine if vulnerabilities can be exploited in your environment, ensuring a focused and relevant risk assessment.

Combine reachability insights (if available/detected) with CVSS, EPSS, KEV, and VEX data to form a comprehensive risk assessment. This ensures that you prioritize vulnerabilities that are not only severe and exploitable but also relevant to your specific threat landscape.

Email chelsea@fossa.com for a FREE pdf on vulnerability resolution!!!!!

Bonus open source compliance resolution ⋆°inside°⋆



Outline

Vulnerability remediation tactics

You've found a vulnerability in a third-party component that your code uses. You're confident that it's really present and exploitable. You've done all your due diligence, and you want to get rid of it. How do you do that?

Generally, if a fix is available, there are two approaches you can take here:

1. You can patch the component in-place.
2. You can upgrade to a component version that does not have the vulnerability, by either:
   a. Pinning the vulnerable component to a fixed version.
   b. Doing iterated component upgrades until the vulnerable component has been removed from your dependency installation plan.

Patching In-Place

If you have a patch available for the version of the component that you use, you can always apply the patch in-place. This usually means taking a diff and applying it directly to the component's code as part of your build, and then building against the patched component.

Pros

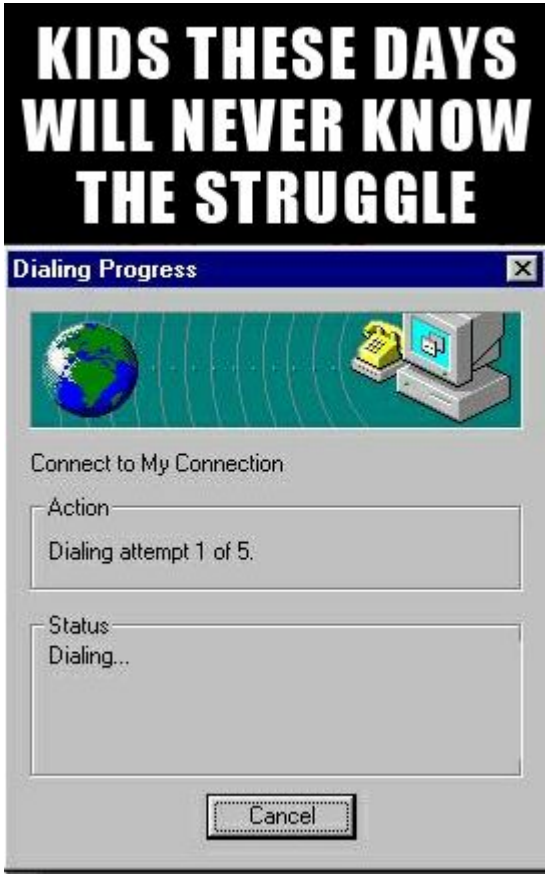
- This generally requires the least amount of application-level changes, as long as the patch roughly preserves the semantics and behavior of the original functionality that you relied on.

Cons

- You need to have a patch available. This is pretty rare, unless you have a dedicated security team that also builds patches.
- Applying patches is finicky. How do you get the patch to persist between builds? Usually this needs some build process hackery, whether that's applying the patch as part of the build or checking in and vendoring the source code of the component, or publishing your own patched variant of the component.
- Patches sometimes require application-level changes anyway, especially if they impact the performance or semantics of the original vulnerable functionality, or if you were relying on implementation details.

Upgrading Components

If you don't have a patch available, you'll need to upgrade to a version of the component that is not impacted by the vulnerability.



QA time!!!

Open to discuss…

- **Share your process around resolving vulnerabilities**
  - **As an OSS maintainer**
  - **You're actually working at a company (startup, enterprise?)**
- **As a *new* open source contributor, how can I help?**

Additional references

CISA.gov justifications:

1. Component_not_present
2. Vulnerable_code_not_present
3. Vulnerable_code_cannot_be_controlled_by_adversary
4. Vulnerable_code_not_in_execute_path
5. Inline_mitigations_already_exist

FOSSA justifications:

1. Component_not_present
2. Incorrect_data_found
3. Inline_mitigations_already_exist
4. Vulnerable_code_cannot_be_controlled_by_adversary
5. Vulnerable_code_not_in_execute_path
6. Vulnerable_code_not_present
7. Other (with space for your text)

CycloneDX's VEX options:

1. code_not_present
2. code_not_reachable
3. requires_configuration
4. requires_dependency
5. requires_environment
6. protected_by_compiler
7. protected_at_runtime
8. protected_at_perimeter
9. protected_by_mitigating_control

Additional references

Learn more about FOSSA:

https://github.com/fossas/fossa-cli/tree/master

https://docs.fossa.com/

https://fossa.com/blog/using-cisa-kev-catalog/