Predicting Security Attacks in FOSS

Why you want it and one way to do it

C.E. Budde   R. Paramitha   F. Massacci
Università di Trento (IT) & Vrije Universiteit (NL)

Vuln4Cast 2023 FIRST Technical Colloquium
Talk overview

1. Introduction
2. Background
3. Forecast model
4. Conclusions
Why You Should Update All Your Software

Updates may sometimes be painful, but they’re necessary to keep your devices and data secure on a dangerous internet.

BY CHRIS HOFFMAN  PUBLISHED AUG 28, 2020

Quick Links

Security Updates 101

What’s the Risk, Really?
Why You Should Update All Your Software

Updates may sometimes be painful, but they're necessary to keep your devices and data secure on a dangerous internet.

By Chris Hoffman  Published Aug 28, 2020
Some motivation (plz!)

- Date: Jul'22 to Dec'22
- Libraries:
  - org.redisson:redisson
    - Versions: 3.17.5, 3.17.6, 3.17.7, 3.18.0, 3.18.1, 3.19.0
  - io.netty:netty-codec
    - Version: 4.1.79, 4.1.80, 4.1.81, 4.1.82, 4.1.83, 4.1.84, 4.1.85, 4.1.86
Some motivation (plz!)

CVE-2022-41915 disclosed!
CVE-2022-41915 affects netty [4.1.83, 4.1.86)
Some motivation (plz!)

CVE-2022-41915 disclosed!
>affects netty [4.1.83, 4.1.86)
Some motivation (plz!)

CVE-2022-41915 disclosed!
- affects netty [4.1.83, 4.1.86)

Forced MOVE
- Correct STAY
- Correct MOVE
- Correct STAY
- Forced MOVE

org.redisson:redisson
- io.netty:netty-codec
Some motivation (plz!)

Hindsight!

CVE-2022-41915 disclosed!

1. **org.redisson:redisson**
   - 3.17.5
   - 3.17.6
   - 3.17.7
   - 3.18.0
   - 3.18.1
   - 3.19.0

2. **io.netty:netty-codec**
   - 4.1.79
   - 4.1.80
   - 4.1.81
   - 4.1.82
   - 4.1.83
   - 4.1.84
   - 4.1.85
   - 4.1.86

Forced MOVE
Wrong MOVE
Correct STAY
Correct MOVE
Correct STAY
Forced MOVE

- Discloses CVE-2022-41915: affects netty [4.1.83, 4.1.86]
Some motivation (plz!)

Hindsight!

<table>
<thead>
<tr>
<th>Time</th>
<th>Package</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jul'22</td>
<td>org.redisson:redisson</td>
<td>3.17.5</td>
<td></td>
</tr>
<tr>
<td>Aug'22</td>
<td>org.redisson:redisson</td>
<td>3.17.6</td>
<td>Correct STAY</td>
</tr>
<tr>
<td>Sep'22</td>
<td>org.redisson:redisson</td>
<td>3.17.7</td>
<td>Correct MOVE</td>
</tr>
<tr>
<td>Oct'22</td>
<td>org.redisson:redisson</td>
<td>3.18.0</td>
<td>Correct STAY</td>
</tr>
<tr>
<td>Nov'22</td>
<td>org.redisson:redisson</td>
<td>3.18.1</td>
<td>Wrong MOVE</td>
</tr>
<tr>
<td>Dec'22</td>
<td>org.redisson:redisson</td>
<td>3.19.0</td>
<td>Forced MOVE</td>
</tr>
<tr>
<td></td>
<td>io.netty:netty-codec</td>
<td>4.1.79</td>
<td>CVE-2022-41915 disclosed!</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.1.80</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.1.81</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.1.82</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.1.83</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.1.84</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.1.85</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.1.86</td>
<td></td>
</tr>
</tbody>
</table>

Forced MOVE
Wrong MOVE
Correct STAY
Correct MOVE

CVE-2022-41915 disclosed!
-> affects netty [4.1.83, 4.1.86]
Some motivation (plz!)

Developer perspective in time:

- org.redisson:redisson
- io.netty:netty-codec

\[ d = 4.179 \]
Some motivation (plz!)

Developer perspective in time:

- org.redisson:redisson
  - 3.17.5

- io.netty:netty-codec
  - 4.1.79

Timeline:
- Jul'22
- Aug'22
- Sep'22
- Oct'22
- Nov'22
- Dec'22
Some motivation (plz!)

Developer perspective in time:

Jul'22  Aug'22  Sep'22  Oct'22  Nov'22  Dec'22
time

org.redisson:redisson
3.17.5

io.netty:netty-codec
4.1.79
4.1.80
Some motivation (plz!)

Developer perspective in time:

- **org.redisson:redisson**: 3.17.5 → 3.17.6
- **io.netty:netty-codec**: 4.1.79 → 4.1.80

Timeline: Sep'22 Aug'22 Jul'22 Dec'22 Oct'22 Nov'22
Some motivation (plz!)

Developer perspective in time:
Some motivation (plz!)

Developer perspective in time:
Some motivation (plz!)

Developer perspective in time:

- org.redisson:redisson
  - 3.17.5
  - 3.17.6
  - 3.17.7
  - 3.18.0
  - 3.18.1

- io.netty:netty-codec
  - 4.1.79
  - 4.1.80
  - 4.1.81
  - 4.1.82
  - 4.1.83
  - 4.1.84
  - 4.1.85

Timeline:
- Jul'22
- Aug'22
- Sep'22
- Oct'22
- Nov'22
- Dec'22
Some motivation (plz!)

Developer perspective in time:

CVE-2022-41915 disclosed!
affects netty [4.1.83, 4.1.86]
Some motivation (plz!)

Developer perspective in time:

- `org.redisson:redisson` 3.17.5
- `io.netty:netty-codec` 4.1.79
- CVE-2022-41915 disclosed!

Forced MOVE
Wrong MOVE
Correct STAY
Correct STAY
Some motivation (plz!)

Developer perspective in time:

Is there a best time to update?
Questions

Q1  How does time affect the $\text{Pr(\text{vuln.})}$?

Q2  Which other factors affect $\text{Pr(\text{vuln.})}$?
Questions

Q1  How does time affect the $Pr(\text{vuln.})$?
    ▷ best time to update?

Q2  Which other factors affect $Pr(\text{vuln.})$?
Questions

Q1  How does time affect the Pr(vuln.)?
    ▷ best time to update?

Q2  Which other factors affect Pr(vuln.)?
    ▷ measurable software metrics
1. Unpublished/Undetected vulnerabilities:
   - we study publication of CVEs;
1. Unpublished/Undetected vulnerabilities:
   • we study publication of CVEs;
   • keep it high-level, no code analysis.
1. Unpublished/Undetected vulnerabilities:
   • we study publication of CVEs;
   • keep it high-level, no code analysis.

2. Probability of exploitation:
   • we study publication of CVEs;
1. Unpublished/Undetected vulnerabilities:
   • we study publication of CVEs;
   • keep it high-level, no code analysis.

2. Probability of exploitation:
   • we study publication of CVEs;
   • … but check the work of the EPSS!
1. Introduction

2. Background

3. Forecast model

4. Conclusions
State of the ART

Models to predict vulnerabilities

<table>
<thead>
<tr>
<th>Work</th>
<th>Goal</th>
<th>Data</th>
<th>Method</th>
<th>Approach</th>
<th>Projects/Libs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[4]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[2]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[16]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[13]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[15]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[14]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[6]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[8]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[23]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[24]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[25]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[21]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[1]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[9]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[3]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[7]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[22]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[18]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[12]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[19]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[17]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[26]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[10]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[20]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Purport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find vulnerabilities regardless of existent logs such as CVEs (although CWEs may be used). This includes formal methods and static/dynamic code analysis.</td>
</tr>
<tr>
<td>Detect known vulnerabilities (and their correlation to developer activity metrics) from VCS only—e.g. commit churn, peer comments, etc.</td>
</tr>
<tr>
<td>Detect known vulnerabilities (and their correlation to code metrics) from code only—e.g. number of classes, code cloning, cyclomatic complexity, etc.</td>
</tr>
<tr>
<td>Detect known vulnerabilities using code or VCS, via dependency-aware models that can find the offending code to help correcting it (own vs. third-party libraries).</td>
</tr>
<tr>
<td>Time regression to predict vulnerabilities from NVD logs, but the models lack data from the security domain.</td>
</tr>
</tbody>
</table>
State of the Art

Models to predict vulnerabilities

<table>
<thead>
<tr>
<th>Work</th>
<th>Goal</th>
<th>Data</th>
<th>Method</th>
<th>Approach</th>
<th>Projects/Libs.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Disc</td>
<td>Pred</td>
<td>CVEs</td>
<td>Code</td>
<td>VCS</td>
</tr>
<tr>
<td>[4]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[2]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[16]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[11]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[13]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[15]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[14]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[6]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[8]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[23]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[24]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[25]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[21]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[1]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[9]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[3]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[7]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[22]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[18]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[12]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[19]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[17]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[26]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[10]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[20]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
# State of the ART

## Models to predict vulnerabilities

<table>
<thead>
<tr>
<th>Work</th>
<th>Goal</th>
<th>Data</th>
<th>Method</th>
<th>Approach</th>
<th>Projects/Libs.</th>
<th>Language</th>
<th>#</th>
<th>Purport</th>
</tr>
</thead>
<tbody>
<tr>
<td>[4]</td>
<td>Disc.</td>
<td>CVEs</td>
<td>Code</td>
<td>Corr.</td>
<td>AH</td>
<td>C</td>
<td>3</td>
<td>Find vulnerabilities regardless of existent logs such as CVEs (although CWEs may be used). This includes formal methods and static/dynamic code analysis.</td>
</tr>
<tr>
<td>[16]</td>
<td></td>
<td>CVEs</td>
<td>VCS</td>
<td></td>
<td>ML</td>
<td>Java</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>[11]</td>
<td>Disc.</td>
<td>CVEs</td>
<td>VCS</td>
<td></td>
<td></td>
<td>C, C++, Java, JS, SQL</td>
<td>10</td>
<td>Detect known vulnerabilities and their correlation to developer activity metrics from VCS only—e.g. commit churn, peer comments, etc.</td>
</tr>
<tr>
<td>[15]</td>
<td></td>
<td>CVEs</td>
<td>VCS</td>
<td></td>
<td></td>
<td>C, ASM</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>[14]</td>
<td></td>
<td></td>
<td>Code</td>
<td></td>
<td></td>
<td>C, ASM</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>[6]</td>
<td>Disc.</td>
<td>CVEs</td>
<td>VCS</td>
<td></td>
<td></td>
<td>C++</td>
<td>3</td>
<td>Detect known vulnerabilities (and their correlation to code metrics) from code only—e.g. number of classes, code cloning, cyclomatic complexity, etc.</td>
</tr>
<tr>
<td>[23]</td>
<td></td>
<td>CVEs</td>
<td>VCS</td>
<td></td>
<td></td>
<td>Java</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>[25]</td>
<td></td>
<td>CVEs</td>
<td>VCS</td>
<td></td>
<td></td>
<td>Java</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>[21]</td>
<td></td>
<td></td>
<td>Code</td>
<td></td>
<td></td>
<td>Java</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td>Disc.</td>
<td>CVEs</td>
<td>VCS</td>
<td></td>
<td></td>
<td>C/C++</td>
<td>&gt;150k</td>
<td>Detect known vulnerabilities (and their corr. to code and developer activity metrics) from both code and VCS, but without considering the effect of dependencies in their propagation.</td>
</tr>
<tr>
<td>[9]</td>
<td>Pred.</td>
<td></td>
<td>Code</td>
<td></td>
<td></td>
<td>C/C++</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>[3]</td>
<td></td>
<td>CVEs</td>
<td>VCS</td>
<td></td>
<td></td>
<td>C/C++</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>[7]</td>
<td></td>
<td></td>
<td>Code</td>
<td></td>
<td></td>
<td>C/C++, Java</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>[22]</td>
<td></td>
<td>CVEs</td>
<td>VCS</td>
<td></td>
<td></td>
<td>C/C++</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>[18]</td>
<td>Disc.</td>
<td>CVEs</td>
<td>VCS</td>
<td></td>
<td></td>
<td>Java</td>
<td>500</td>
<td>Detect known vulnerabilities using code or VCS, via dependency-aware models that can find the offending code to help correcting it (own vs. third-party libraries).</td>
</tr>
<tr>
<td>[12]</td>
<td>Pred.</td>
<td></td>
<td>Code</td>
<td></td>
<td></td>
<td>Java</td>
<td>&gt;300k</td>
<td></td>
</tr>
<tr>
<td>[19]</td>
<td></td>
<td>CVEs</td>
<td>VCS</td>
<td></td>
<td></td>
<td>Java, Ruby, Python</td>
<td>450</td>
<td></td>
</tr>
<tr>
<td>[17]</td>
<td></td>
<td></td>
<td>Code</td>
<td></td>
<td></td>
<td>Java</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>[26]</td>
<td></td>
<td>CVEs</td>
<td>VCS</td>
<td></td>
<td></td>
<td>Agnostic</td>
<td>9</td>
<td>Time regression to predict vulnerabilities from NVD logs, but the models lack data from the security domain.</td>
</tr>
<tr>
<td>[20]</td>
<td></td>
<td>CVEs</td>
<td>VCS</td>
<td></td>
<td></td>
<td>Agnostic</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
## State of the Art

### Models to predict vulnerabilities

<table>
<thead>
<tr>
<th>Work</th>
<th>Goal</th>
<th>Data</th>
<th>Method</th>
<th>Approach</th>
<th>Projects/Libs.</th>
<th>Purport</th>
</tr>
</thead>
<tbody>
<tr>
<td>[4]</td>
<td>Disc. Pred.</td>
<td>CVEs, Code, VCS</td>
<td>Corr.</td>
<td>AH, SA, ML</td>
<td>C</td>
<td>Find vulnerabilities regardless of existent logs such as CVEs (although CWEs may be used). This includes formal methods and static/dynamic code analysis.</td>
</tr>
<tr>
<td>[11]</td>
<td>Disc. Pred.</td>
<td>CVEs, Code, VCS</td>
<td>Corr.</td>
<td>AH, SA, ML</td>
<td>C</td>
<td>Detect known vulnerabilities (and their correlation to developer activity metrics) from VCS only—e.g. commit churn, peer comments, etc.</td>
</tr>
</tbody>
</table>
Models to predict vulnerabilities

<table>
<thead>
<tr>
<th>Work</th>
<th>Goal</th>
<th>Data</th>
<th>Method</th>
<th>Approach</th>
<th>Projects/Libs.</th>
<th>Purport</th>
</tr>
</thead>
<tbody>
<tr>
<td>[4]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[2]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[16]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[11]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[13]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[15]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[14]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[6]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[8]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[23]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[24]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[25]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[21]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[1]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[9]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[3]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[7]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[22]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[18]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[12]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[19]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[17]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[26]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[10]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[20]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
State of the ART

Models to predict vulnerabilities

**Q2** \( \text{Pr}(\text{vuln.}) \) as function of **software metrics**

**Q1** \( \text{Pr}(\text{vuln.}) \) as function of **time**
Q2  Pr(vuln.) as function of software metrics

- ML & statistical analysis to correlate SE metrics to existent vulnerabilities

Q1  Pr(vuln.) as function of time
Q2  \( \Pr(\text{vuln.}) \) as function of **software metrics**

- ML & statistical analysis to correlate SE metrics to existent vulnerabilities
- human-in-the-loop metrics, including VCS (#commits, seniority…) 

Q1  \( \Pr(\text{vuln.}) \) as function of **time**
Q2  $\Pr(\text{vuln.})$ as function of software metrics

- ML & statistical analysis to correlate SE metrics to existent vulnerabilities
- human-in-the-loop metrics, including VCS (#commits, seniority…)
- (a few) considerations of own and 3$^{rd}$ party dependencies

Q1  $\Pr(\text{vuln.})$ as function of time
Q2  $\Pr(\text{vuln.})$ as function of **software metrics**

- ML & statistical analysis to correlate SE metrics to existent vulnerabilities
- human-in-the-loop metrics, including VCS (#commits, seniority…)
- (a few) considerations of own and 3rd party dependencies

Q1  $\Pr(\text{vuln.})$ as function of **time**

- time-regression models on CVE publications ($\approx$ FinTech)
• Studies typically try to detect, not foretell vulnerabilities.
Gap analysis

• Studies typically try to detect, not foretell vulnerabilities.

• The dependency tree is seldom analysed (own code only).
• Studies typically try to *detect*, not *foretell* vulnerabilities.

• The dependency tree is seldom analysed (own code only).

• The rare-event nature of vulnerabilities is disregarded.
Gap analysis

• Studies typically try to detect, not foretell vulnerabilities.

• The dependency tree is seldom analysed (own code only).

• The rare-event nature of vulnerabilities is disregarded.

We propose white-box model(s) to fill these gaps
Forecast model

1. Introduction

2. Background

3. Forecast model

4. Conclusions

Time Dependency Trees

CVE root-lib PDFs
Time Dependency Trees

Dependency Trees in time

\[ D(\ell_{a_1}) : \]

\[ \ell_{a_1} \]

\[ \ell_{d_2} \quad \ell_{c_1} \]

\[ \ell_{d_1} \]
Time Dependency Trees

Dependency Trees in time

\[ D(\ell_{a1}) \]

\[ D(\ell_{a2}) \]

\[ \ell_{a1} \rightarrow \ell_{d2} \rightarrow \ell_{d1} \]

\[ \ell_{c1} \]

\[ \ell_{d3} \rightarrow \ell_{c1} \]

\[ \ell_{a2} \rightarrow \ell_{d1} \]
Time Dependency Trees

Dependency Trees in time

$D(\ell_{a1})$: $\ell_{a1}$

$D(\ell_{a2})$: $\ell_{a2}$

$D(\ell_{a3})$: $\ell_{a3}$

$\ell_{d2}$

$\ell_{c1}$

$\ell_{d3}$

$\ell_{c1}$

$\ell_{d1}$

$\ell_{c2}$

$\ell_{d1}$
Time Dependency Trees

Dependency Trees in time

\[ \{D(\ell_{a_i})\}_{i=1}^{\ell} \]
Dependency Trees in time

\[ \{ D(\ell_{a_i}) \}_{i=1}^{3} \]:

Time Dependency Tree

\[ D_T(\ell_a) : \]
Time Dependency Trees

Dependency Trees in time

\[ \{D(l_{a_i})\}_{i=1}^{3} : \]

Time Dependency Tree

\[ D_T(l_a): \]

Main library \((l_a)\)
Time Dependency Trees

Dependency Trees in time

\[ \{D(\ell_{a_i})\}_{i=1}^3: \]

Time Dependency Tree

\[ DT(\ell_a): \]

Main library (\(\ell_a\))

Time span (\(T\))
Time Dependency Trees

Dependency Trees in time

\{D(l_{a_i})\}_{i=1}^{3}:

\[ D_T(l_a) : \]

\[ D_t(l_a) = D(l_{a_1}) \]

for any time point \( t \in T \)

after the release of \( l_{a_1} \) and

before the release of \( l_{a_2} \)
Properties of TDT $D_T(\ell)$

- Minimal graph representation (no lib-version repetition)
Properties of TDT $D_T(\ell)$

- Minimal graph representation (no lib-version repetition)
- Canonical for library $\ell$ and time span $T$
Properties of TDT $D_T(\ell)$

- Minimal graph representation (no lib-version repetition)
- Canonical for library $\ell$ and time span $T$
- Natural lifting of dependency trees to time
Properties of TDT $D_T(\ell)$

Theoretical

- Minimal graph representation (no lib-version repetition)
- Canonical for library $\ell$ and time span $T$
- Natural lifting of dependency trees to time
Properties of TDT $D_T(\ell)$

**Theoretical**

- Minimal graph representation (no lib-version repetition)
- Canonical for library $\ell$ and time span $T$
- Natural lifting of dependency trees to time

**Practical**

- Time-indexing $D_t(\ell)$ yields the dep. tree at time $t \in T$
Properties of TDT $D_T(\ell)$

Theoretical

- Minimal graph representation (no lib-version repetition)
- Canonical for library $\ell$ and time span $T$
- Natural lifting of dependency trees to time

Practical

- Time-indexing $D_t(\ell)$ yields the dep. tree at time $t \in T$
- Library-slicing $D_T(\ell)|_d$ yields all instances of dependency $d$ during time $T$
Properties of TDT $D_T(\ell)$

Theoretical

• Minimal graph representation (no lib-version repetition)
• Canonical for library $\ell$ and time span $T$
• Natural lifting of dependency trees to time

Practical

• Time-indexing $D_t(\ell)$ yields the dep. tree at time $t \in T$
• Library-slicing $D_T(\ell)|_d$ yields all instances of dependency $d$ during time $T$
• Reachability analysis can spot single-points-of-failure
SPoF in time and dependencies

My personal project uses $\ell_{1.0}$

---

Graph showing dependencies:

- $\ell_{0.8} \rightarrow \ell_{0.9} \rightarrow \ell_{1.0} \rightarrow \ell_{1.1}$
- $x_{3.0} \rightarrow x_{3.3}$
- $y_{5.0.0} \rightarrow y_{5.0.1} \rightarrow y_{5.8.3}$
- $z_{2.0} \rightarrow z_{2.1}$
- $z_{2.2}$
My personal project uses $\ell_{1.0}$
My personal project uses $\ell_{1.0}$

Should I downgrade to $\ell_{0.9}$ or upgrade to $\ell_{1.1}$?
Properties of TDT $D_T(\ell)$

Theoretical

- Minimal graph representation (no lib-version repetition)
- Canonical for library $\ell$ and time span $T$
- Natural lifting of dependency trees to time

Practical

- Time-indexing $D_t(\ell)$ yields the dep. tree at time $t \in T$
- Library-slicing $D_T(\ell)|_d$ yields all instances of dependency $d$ during time $T$
- Reachability analysis can spot single-points-of-failure
Properties of TDT $D_T(\ell)$

**Theoretical**

- Minimal graph representation (no lib-version repetition)
- Canonical for library $\ell$ and time span $T$
- Natural lifting of dependency trees to time

**Practical**

- Time-indexing $D_t(\ell)$ yields the dep. tree at time $t \in T$
- Library-slicing $D_T(\ell)\big|_d$ yields all instances of dependency $d$ during time $T$
- Reachability analysis can spot single-points-of-failure
- Can measure health/risk of development environment
Forecast model

1. Introduction
2. Background
3. Forecast model
4. Conclusions
Publication of CVE since time of code release

- org.redisson:redisson
  - 3.17.5
  - 3.17.6
  - 3.17.7
  - 3.18.0
  - 3.18.1

- io.netty:netty-codec
  - 4.1.79
  - 4.1.80
  - 4.1.81
  - 4.1.82
  - 4.1.83
  - 4.1.84
  - 4.1.85
  - 4.1.86

CVE-2022-41915 disclosed!
affects netty [4.1.83, 4.1.86)
Publication of CVE since time of code release

CVE-2022-41915 disclosed!
affects netty [4.1.83, 4.1.86)
Publication of CVE since time of code release

org.redisson:redisson

3.17.5
ℓa ℓa
3.17.6
3.17.5
ℓa
3.17.7
3.18.0
3.18.1

io.netty:netty-codec

4.1.79
4.1.80
4.1.81
4.1.82
4.1.83
4.1.84
4.1.85
4.1.86

Jul'22 Aug'22 Sep'22 Oct'22 Nov'22 Dec'22
time

CVE-2022-41915 disclosed!
affects netty [4.1.83, 4.1.86]

18/34
Publication of CVE since time of code release

CVE-2022-41915 disclosed!
affects netty [4.1.83, 4.1.86]
Publication of CVE since time of code release

CVE-2022-41915 disclosed!
affects netty [4.1.83, 4.1.86]
Publication of CVE since time of code release

CVE-2022-41915 disclosed!
affects netty [4.1.83, 4.1.86)

1M 2M 3M 4M ...

0 2.0e-03 4.0e-03 6.0e-03 8.0e-03 1.0e-02

Time since lib. release

Probability of CVE public.

CVE-2022-41915 disclosed!
affects netty [4.1.83, 4.1.86)
Publication of CVE since time of code release

** CVE-2022-41915 disclosed! 
- **affects** netty [4.1.83, 4.1.86)
Count each CVE as one data point
  • must choose one affected version!
Rules of the game

- Count each CVE as one data point
  - must choose one affected version!
Rules of the game

- Count each CVE as one data point
  - must **choose one** affected version!
Rules of the game

- Count each CVE as one data point
  - must choose one affected version!

- Discriminate per development environment
  - e.g. Java and C/C++ have different vuln. (and times!)
Rules of the game

- Count each CVE as one data point
  - must choose one affected version!

- Discriminate per development environment
  - e.g. Java and C/C++ have different vuln. (and times!)
Rules of the game

- Count each CVE as one data point
  - must choose one affected version!

- Discriminate per development environment
  - e.g. Java and C/C++ have different vuln. (and times!)

- Discriminate per library type
  - consider security-relevant code metrics
Rules of the game

- Count each CVE as one data point
  - must choose one affected version!

- Discriminate per development environment
  - e.g. Java and C/C++ have different vuln. (and times!)

- Discriminate per library type
  - consider security-relevant code metrics
Security-relevant code metrics

CVEs with the 'Java' keyword

- Physical: 6
- Local: 199
- Adjacent: 11
- Network: 4321

Total CVEs from the NVD: 4537
Security-relevant code metrics

Used in remote networks

CVEs with the 'Java' keyword

<table>
<thead>
<tr>
<th>Attack Vector</th>
<th># CVEs from the NVD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical</td>
<td>6</td>
</tr>
<tr>
<td>Local</td>
<td>199</td>
</tr>
<tr>
<td>Adjacent</td>
<td>11</td>
</tr>
<tr>
<td>Network</td>
<td>4321</td>
</tr>
</tbody>
</table>
Security-relevant code metrics
Security-relevant code metrics

(Own) Code size
## Security-relevant code metrics

<table>
<thead>
<tr>
<th>Work</th>
<th>Goal</th>
<th>Data</th>
<th>Method</th>
<th>Approach</th>
<th>Projects/Libs.</th>
<th>Purport</th>
</tr>
</thead>
<tbody>
<tr>
<td>[4]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>C</td>
<td>3</td>
</tr>
<tr>
<td>[2]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>PHP</td>
<td>3</td>
</tr>
<tr>
<td>[16]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Java</td>
<td>4</td>
</tr>
<tr>
<td>[11]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>C</td>
<td>3</td>
</tr>
<tr>
<td>[13]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>[15]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>C, ASM</td>
<td>3</td>
</tr>
<tr>
<td>[14]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>C, ASM</td>
<td>1</td>
</tr>
<tr>
<td>[6]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>C/C++</td>
<td>3</td>
</tr>
<tr>
<td>[8]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Java</td>
<td>7</td>
</tr>
<tr>
<td>[23]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Java</td>
<td>4</td>
</tr>
<tr>
<td>[24]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Java</td>
<td>3</td>
</tr>
<tr>
<td>[25]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Java</td>
<td>5</td>
</tr>
<tr>
<td>[21]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>C</td>
<td>7</td>
</tr>
<tr>
<td>[1]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>C/C++</td>
<td>&gt;150k</td>
</tr>
<tr>
<td>[9]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>C/C++</td>
<td>8</td>
</tr>
<tr>
<td>[3]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>C/C++</td>
<td>5</td>
</tr>
<tr>
<td>[7]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>C/C++, Java</td>
<td>1</td>
</tr>
<tr>
<td>[22]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>C/C++</td>
<td>2</td>
</tr>
<tr>
<td>[18]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Java</td>
<td>500</td>
</tr>
<tr>
<td>[12]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Java</td>
<td>&gt;300k</td>
</tr>
<tr>
<td>[19]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Java, Ruby, Python</td>
<td>450</td>
</tr>
<tr>
<td>[17]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Java</td>
<td>200</td>
</tr>
<tr>
<td>[26]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Agnostic</td>
<td>9</td>
</tr>
<tr>
<td>[10]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Agnostic</td>
<td>25</td>
</tr>
<tr>
<td>[20]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Agnostic</td>
<td>5</td>
</tr>
</tbody>
</table>
Security-relevant code metrics

Used in remote networks

Own-code size
Security-relevant code metrics

Own-code size

Used in remote networks

My favourite correlation

My favourite metric A vs. My favourite metric B
Security-relevant code metrics

Own-code size

My favourite correlation

Used in remote networks
Security-relevant code metrics

Used in remote networks

My favourite correlation

Own-code size
Security-relevant code metrics

My favourite correlation

Own-code size
Security-relevant code metrics

My favourite correlation

Used in remote networks

Own-code size
Security-relevant code metrics

My favourite correlation

Used in remote networks

Own-code size

My favourite metric A

My favourite metric B

Time since lib. release

Probability of CVE public.
Security-relevant code metrics

Used in remote networks

Own-code size
On overfitting and rare events

- Count each CVE as one data point
- Discriminate per development environment
- Discriminate per library type
- Clusterisation mustn’t be too thin
  - Few divisions per metric-dimension
  - Few metric-dimensions
On overfitting and rare events

My favourite correlation

- Count each CVE as one data point
- Discriminate per development environment
- Discriminate per library type
- Clusterisation mustn’t be too thin
  - few divisions per metric-dimension
  - few metric-dimensions
On overfitting and rare events
On overfitting and rare events

- Count each CVE as one data point
- Discriminate per development environment
- Discriminate per library type
On overfitting and rare events

- Count each CVE as one data point
- Discriminate per development environment
- Discriminate per library type
- Clusterisation mustn’t be too thin
  - few divisions per metric-dimension
  - few metric-dimensions
Enough!

Gimme results
Here ya go
Q1  $\Pr(\text{vuln.})$ as function of time

Q2  $\Pr(\text{vuln.})$ as function of software metrics
Survival analysis on library update

- **org.redisson:redisson**
  - 3.17.5
  - 3.17.6

- **io.netty:netty-codec**
  - 4.1.79
  - 4.1.80
  - 4.1.81
  - 4.1.82
Survival analysis on library update

- org.redisson:redisson
  - 3.17.5
  - 3.17.6
- io.netty:netty-codec
  - 4.1.79
  - 4.1.80
  - 4.1.81
  - 4.1.82
Survival analysis on library update

\( A \xrightarrow{t} B \) means that we change from dependency \( \ell_A \) to \( \ell_B \) in \( t \) time units counting from \( t_0 \) (“today”).

- \( \ell_A \) was released on \( t_A < t_0 \), \( \ell_B \) on \( t_B < t_0 \), \( t_A \succ t_B \)

\[
\Pr_{A,B}(t) = 1 - SF_{A,t} + \Delta t_{A} \cdot CDF_{B,t} + \Delta t_{B}
\]

where \( \Delta t_x = |t_x - t_0| \).
Survival analysis on library update

\[ A^t \rightarrow B \] means that we change from dependency \( \ell_A \) to \( \ell_B \) in \( t \) time units counting from \( t_0 \) (“today”).

▷ \( \ell_A \) was released on \( t_A < t_0 \), \( \ell_B \) on \( t_B < t_0 \), \( t_A \gg t_B \)

Q: \( Pr_{A,B}(t) = \) probability of vuln. of \( A \rightarrow B \) as a function of \( t \)
Survival analysis on library update

$A \xrightarrow{t} B$ means that we change from dependency $\ell_A$ to $\ell_B$ in $t$ time units counting from $t_0$ (“today”).

$\triangleright \ell_A$ was released on $t_A < t_0$, $\ell_B$ on $t_B < t_0$, $t_A \bowtie t_B$

Q: $\Pr_{A,B}(t) =$ probability of vuln. of $A \xrightarrow{t} B$ as a function of $t$

A: $\Pr_{A,B}(t) = 1 - SF_A \left(t + \Delta t_A\right) \cdot \text{CDF}_B \left(t + \Delta t_B\right)$ where $\Delta t_x \doteq |t_x - t_0|$
A $t \rightarrow B$ means that we change from dependency $\ell_A$ to $\ell_B$ in $t$ time units counting from $t_0$ (“today”).

$\triangleright \ell_A$ was released on $t_A < t_0$, $\ell_B$ on $t_B < t_0$, $t_A \nless t_B$

**Q:** $Pr_{A,B}(t) =$ probability of vuln. of $A \stackrel{t}{\rightarrow} B$ as a function of $t$

**A:** $Pr_{A,B}(t) = 1 - SF_A(t + \Delta t_A) \cdot CDF_B(t + \Delta t_B)$ where $\Delta t_x = |t_x - t_0|

- vuln. in $\ell_A$ before change
- vuln. in $\ell_B$ after change
Survival analysis on library update

$A \xrightarrow{t} B$ means that we change from dependency $\ell_A$ to $\ell_B$ in $t$ time units counting from $t_0$ (“today”).

- $\ell_A$ was released on $t_A < t_0$, $\ell_B$ on $t_B < t_0$, $t_A \gg t_B$

**Q:** $\Pr_{A,B}(t) =$ probability of vuln. of $A \xrightarrow{t} B$ as a function of $t$

**A:** $\Pr_{A,B}(t) = 1 - \text{SF}_A(t + \Delta t_A) \cdot \text{CDF}_B(t + \Delta t_B)$ where $\Delta t_x \doteq |t_x - t_0|$

- vuln. in $\ell_A$ before change
- vuln. in $\ell_B$ after change
Survival analysis on library update

$A \xrightarrow{t} B$ means that we change from dependency $\ell_A$ to $\ell_B$ in $t$ time units counting from $t_0$ (“today”).

$\triangleright \ell_A$ was released on $t_A < t_0$, $\ell_B$ on $t_B < t_0$, $t_A \gg t_B$

**Q:** $\Pr_{A,B}(t) =$ probability of vuln. of $A \xrightarrow{t} B$ as a function of $t$

**A:**

$\Pr_{A,B}(t) = 1 - SF_A(t + \Delta t_A) \ CDF_B(t + \Delta t_B)$  \hspace{1cm} \text{where} \hspace{0.5cm} \Delta t_x = |t_x - t_0|$

$\ell_A$ released $184$ days ago

$\ell_B$ released $21$ days ago

$t_A = 184 \text{ days}$

$t_B = 21 \text{ days}$
Survival analysis on library update

\( A \xrightarrow{t} B \) means that we change from dependency \( \ell_A \) to \( \ell_B \) in \( t \) time units counting from \( t_0 \) (“today”).

\( \triangleright \ell_A \) was released on \( t_A < t_0, \ell_B \) on \( t_B < t_0, t_A \gg t_B \)

**Q:** \( \Pr_{A,B}(t) = \) probability of vuln. of \( A \xrightarrow{t} B \) as a function of \( t \)

**A:** \( \Pr_{A,B}(t) = 1 - \text{SF}_A(t + \Delta t_A) \cdot \text{CDF}_B(t + \Delta t_B) \) where \( \Delta t_x = |t_x - t_0| \)

\( t_A = 17 \text{ days} \)
\( t_B = 85 \text{ days} \)

\( t_A = 184 \text{ days} \)
\( t_B = 21 \text{ days} \)
**Survival analysis on library update**

\( A \xrightarrow{t} B \) means that we change from dependency \( \ell_A \) to \( \ell_B \) in \( t \) time units counting from \( t_0 \) ("today").

\( \triangleright \) \( \ell_A \) was released on \( t_A < t_0, \ell_B \) on \( t_B < t_0, t_A \gg t_B \)

**Q:** \( \Pr_{A,B}(t) = \) probability of vuln. of \( A \xrightarrow{t} B \) as a function of \( t \)

**A:** \( \Pr_{A,B}(t) = 1 - SF_A(t + \Delta t_A) \cdot CDF_B(t + \Delta t_B) \) where \( \Delta t_x \equiv |t_x - t_0| \)

---

**Prob. of vuln. when changing \( t_A \to t_B \) at time \( T \)**

\( t_A = 17 \) days

\( t_B = 85 \) days

**Prob. of vuln. when changing \( t_A \to t_B \) at time \( T \)**

\( t_A = 184 \) days

\( t_B = 21 \) days

---

---

---
Survival analysis on library update

$A \xrightarrow{t} B$ means that we change from dependency $\ell_A$ to $\ell_B$ in $t$ time units counting from $t_0$ (“today”).

- $\ell_A$ was released on $t_A < t_0$, $\ell_B$ on $t_B < t_0$, $t_A \bowtie t_B$

**Q:** $\Pr_{A,B}(t) =$ probability of vuln. of $A \xrightarrow{t} B$ as a function of $t$

**A:** $\Pr_{A,B}(t) = 1 - SF_A(t + \Delta t_A) \cdot CDF_B(t + \Delta t_B)$ where $\Delta t_x = |t_x - t_0|$

- $t_A = 17$ days
- $t_B = 85$ days

$t_A = 184$ days
- $t_B = 21$ days
Survival analysis on library update

$A \xrightarrow{t} B$ means that we change from dependency $\ell_A$ to $\ell_B$ in $t$ time units counting from $t_0$ ("today").

$\triangleright\ell_A$ was released on $t_A < t_0$, $\ell_B$ on $t_B < t_0$, $t_A \gg t_B$

**Q:** $Pr_{A,B}(t) =$ probability of vuln. of $A \xrightarrow{t} B$ as a function of $t$

**A:** $Pr_{A,B}(t) = 1 - SF_A(t + \Delta t_A) \cdot CDF_B(t + \Delta t_B)$ where $\Delta t_x = |t_x - t_0|$

$t_A = 17$ days
$t_B = 85$ days

$t_A = 184$ days
$t_B = 21$ days
Survival analysis on library update

\( A \xrightarrow{t} B \) means that we change from dependency \( \ell_A \) to \( \ell_B \) in \( t \) time units counting from \( t_0 \) (“today”).

\( \triangleright \) \( \ell_A \) was released on \( t_A < t_0 \), \( \ell_B \) on \( t_B < t_0 \), \( t_A \gg t_B \)

**Q:** \( \text{Pr}_{A,B}(t) = \text{probability of vuln. of } A \xrightarrow{t} B \text{ as a function of } t \)

**A:** \( \text{Pr}_{A,B}(t) = 1 - \text{SF}_A(t + \Delta t_A) \text{CDF}_B(t + \Delta t_B) \) where \( \Delta t_x \equiv |t_x - t_0| \)

\( t_A = 17 \text{ days} \)
\( t_B = 85 \text{ days} \)

\( t_A = 184 \text{ days} \)
\( t_B = 21 \text{ days} \)
Vulnerabilities from any dependency

Q: $\Pr_{A,B}(t) = \text{probability of vuln. in } \ell_A \text{ or } \ell_B \text{ before } t$
Vulnerabilities from any dependency

Q: $\Pr_{A,B}(t) = \text{probability of vuln. in } \ell_A \text{ or } \ell_B \text{ before } t$

A: $\Pr_{A,B}(t) = \Pr(\min(\ell_A, \ell_B) \leq t) = 1 - (1 - \Pr_A(t))(1 - \Pr_B(t))$
Vulnerabilities from any dependency

**Q:** $\Pr_{A,B}(t) = \text{probability of vuln. in } \ell_A \text{ or } \ell_B \text{ before } t$

**A:** $\Pr_{A,B}(t) = \Pr(\min(\ell_A, \ell_B) \leq t) = 1 - (1 - \Pr_A(t))(1 - \Pr_B(t))$
Vulnerabilities from any dependency

**Q:** \( \Pr_{A,B}(t) = \text{probability of vuln. in } \ell_A \text{ or } \ell_B \text{ before } t \)

**A:** \( \Pr_{A,B}(t) = \Pr(\min(\ell_A, \ell_B) \leq t) = 1 - (1 - \Pr_A(t))(1 - \Pr_B(t)) \)
**Q:** \( \Pr_{A,B}(t) = \text{probability of vuln. in } \ell_A \text{ or } \ell_B \text{ before } t \)

**A:** \( \Pr_{A,B}(t) = \Pr(\min(\ell_A, \ell_B) \leq t) = 1 - (1 - \Pr_A(t))(1 - \Pr_B(t)) \)

\[ t_A = 123 \text{ days} \]
\[ t_B = 14 \text{ days} \]
Vulnerabilities from any dependency

Q: $\Pr_{A,B}(t) = \text{probability of vuln. in } \ell_A \text{ or } \ell_B \text{ before } t$

A: $\Pr_{A,B}(t) = \Pr(\min(\ell_A, \ell_B) \leq t) = 1 - (1 - \Pr_A(t))(1 - \Pr_B(t))$

$t_A = 28 \text{ days}$
$t_B = 60 \text{ days}$
Vulnerabilities from any dependency

Q: $\Pr_{A,B}(t) = \text{probability of vuln. in } \ell_A \text{ or } \ell_B \text{ before } t$

A: $\Pr_{A,B}(t) = \Pr(\min(\ell_A, \ell_B) \leq t) = 1 - (1 - \Pr_A(t))(1 - \Pr_B(t))$

$t_A = 28 \text{ days}$

$t_B = 60 \text{ days}$

$t_A = 123 \text{ days}$

$t_B = 14 \text{ days}$
Q: \( \Pr_{A,B}(t) = \text{probability of vuln. in } \ell_A \text{ or } \ell_B \text{ before } t \)

A: \( \Pr_{A,B}(t) = \Pr(\min(\ell_A, \ell_B) \leq t) = 1 - (1 - \Pr_A(t))(1 - \Pr_B(t)) \)
Vulnerabilities from any dependency

**Q:** $\Pr_{A,B}(t) = \text{probability of vuln. in } \ell_A \text{ or } \ell_B \text{ before } t$

**A:** $\Pr_{A,B}(t) = \Pr(\min(\ell_A, \ell_B) \leq t) = 1 - (1 - \Pr_A(t))(1 - \Pr_B(t))$

Nice for 2 dependencies...

I have 2000
Vulnerabilities from any dependency

Q: $\Pr_{A,B}(t) = \text{probability of vuln. in } \ell_A \text{ or } \ell_B \text{ before } t$

A: $\Pr_{A,B}(t) = \Pr(\min(\ell_A, \ell_B) \leq t) = 1 - (1 - \Pr_A(t))(1 - \Pr_B(t))$

Nice for 2 dependencies...

I have 2000 TDTs!
Forecast model

1. Introduction
2. Background
3. Forecast model
4. Conclusions
Some things done

- Time Dependency Trees
Some things done

- Time Dependency Trees
  - Aggregate dependency and code-evolution data
Some things done

- Time Dependency Trees
  - Aggregate dependency and code-evolution data
  - Minimal representation with nice properties
Some things done

- Time Dependency Trees
  - Aggregate dependency and code-evolution data
  - Minimal representation with nice properties
  - Framework for large-scale project analysis
Some things done

- **Time Dependency Trees**
  - Aggregate dependency and code-evolution data
  - Minimal representation with nice properties
  - Framework for large-scale project analysis

- **Probability of vulnerabilities as a function of time**
Some things done

- **Time Dependency Trees**
  - Aggregate dependency and code-evolution data
  - Minimal representation with nice properties
  - Framework for large-scale project analysis

- **Probability of vulnerabilities as a function of time**
  - Express time from library release to CVE publication
Some things done

- **Time Dependency Trees**
  - Aggregate dependency and code-evolution data
  - Minimal representation with nice properties
  - Framework for large-scale project analysis

- **Probability of vulnerabilities as a function of time**
  - Express time from library release to CVE publication
  - Discriminate per type of library (security-relevant props.)
Some things done

- Time Dependency Trees
  - Aggregate dependency and code-evolution data
  - Minimal representation with nice properties
  - Framework for large-scale project analysis

- Probability of vulnerabilities as a function of time
  - Express time from library release to CVE publication
  - Discriminate per type of library (security-relevant props.)
  - Base information for probability forecasting
Some things done

- Other metrics to clusterise libraries for PDF-fitting
- Validate in other languages (all Java so far)
- SPoF detection—across versions—in Java/Maven
- c-chains pollution by CVE
Some things done

- to be

- Other metrics to clusterise libraries for PDF-fitting
- Validate in other languages (all Java so far)
- SPoF detection—across versions—in Java/Maven
- c-chains pollution by CVE
Some things done

to be

- Other metrics to clusterise libraries for PDF-fitting
Some things done

- Other metrics to clusterise libraries for PDF-fitting
- Validate in other languages (all Java so far)
Some things done

- Other metrics to clusterise libraries for PDF-fitting
- Validate in other languages (all Java so far)
- SPoF detection—across versions—in Java/Maven
Some things done

to be

- Other metrics to clusterise libraries for PDF-fitting
- Validate in other languages (all Java so far)
- SPoF detection—across versions—in Java/Maven
- c-chains polution by CVE
Questions?
J. Akram and P. Luo.  
**SQVDT: A scalable quantitative vulnerability detection technique for source code security assessment.**  

M. Alohaly and H. Takabi.  
**When do changes induce software vulnerabilities?**  

H. Alves, B. Fonseca, and N. Antunes.  
**Software metrics and security vulnerabilities: Dataset and exploratory study.**  

**Vulnerability prediction from source code using machine learning.**  
*IEEE Access, 8:150672–150684, 2020.*

A. Bosu, J. C. Carver, M. Hafiz, P. Hilley, and D. Janni.  
**Identifying the characteristics of vulnerable code changes: An empirical study.**  
S. Chakraborty, R. Krishna, Y. Ding, and B. Ray.  
Deep learning based vulnerability detection: Are we there yet.  

I. Chowdhury and M. Zulkernine.  
Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities.  

S. Ganesh, T. Ohlsson, and F. Palma.  
Predicting security vulnerabilities using source code metrics.  

S. Kim, S. Woo, H. Lee, and H. Oh.  
VUDDY: A scalable approach for vulnerable code clone discovery.  

D. Last.  
Forecasting zero-day vulnerabilities.  
H. Li, H. Kwon, J. Kwon, and H. Lee.  
**A scalable approach for vulnerability discovery based on security patches.**  

Q. Li, J. Song, D. Tan, H. Wang, and J. Liu.  
**PDGraph: A large-scale empirical study on project dependency of security vulnerabilities.**  

**When a patch goes bad: Exploring the properties of vulnerability-contributing commits.**  

A. Meneely and L. Williams.  
**Secure open source collaboration: An empirical study of Linus’ law.**  
A. Meneely and L. Williams.  
**Strengthening the empirical analysis of the relationship between Linus’ law and software security.**  

Y. Pang, X. Xue, and A. S. Namin.  
**Predicting vulnerable software components through N-gram analysis and statistical feature selection.**  

I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci.  
**Vulnerable open source dependencies: Counting those that matter.**  

I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci.  
**Vuln4Real: A methodology for counting actually vulnerable dependencies.**  
G. A. A. Prana, A. Sharma, L. K. Shar, D. Foo, A. E. Santosa, A. Sharma, and D. Lo. 
Out of sight, out of mind? how vulnerable dependencies affect open-source projects. 

Y. Roumani, J. K. Nwankpa, and Y. F. Roumani. 
**Time series modeling of vulnerabilities.** 

N. Shahmehri, A. Mammar, E. Montes de Oca, D. Byers, A. Cavalli, S. Ardi, and W. Jimenez. 
**An advanced approach for modeling and detecting software vulnerabilities.** 

Y. Shin, A. Meneely, L. Williams, and J. A. Osborne. 
**Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities.** 
Using software metrics for predicting vulnerable classes and methods in Java projects: A machine learning approach.  

Correlation analysis among Java nano-patterns and software vulnerabilities.  

K. Z. Sultana and B. J. Williams. 
Evaluating micro patterns and software metrics in vulnerability prediction.  

Forecasting IT security vulnerabilities – an empirical analysis.  
Predicting Security Attacks in FOSS
Why you want it and one way to do it

C.E. Budde    R. Paramitha    F. Massacci
Università di Trento (IT) & Vrije Universiteit (NL)

Vuln4Cast 2023 FIRST Technical Colloquium