# Forecasting software vulnerabilities

## Probability Density Functions and Time Dependency Trees

C.E. Budde R. Paramitha F. Massacci

14th March 2024

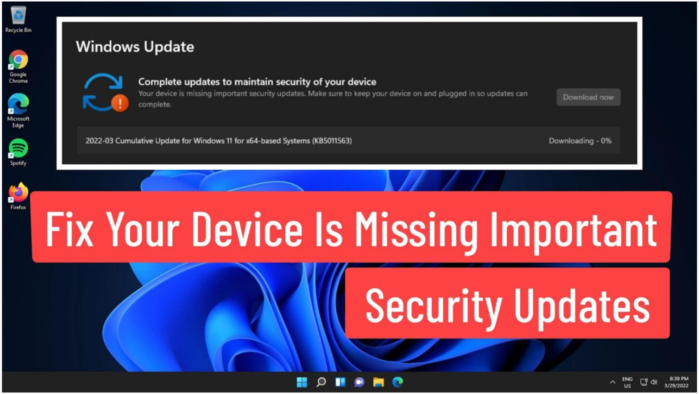
ProSVED final event symposium

# Talk overview

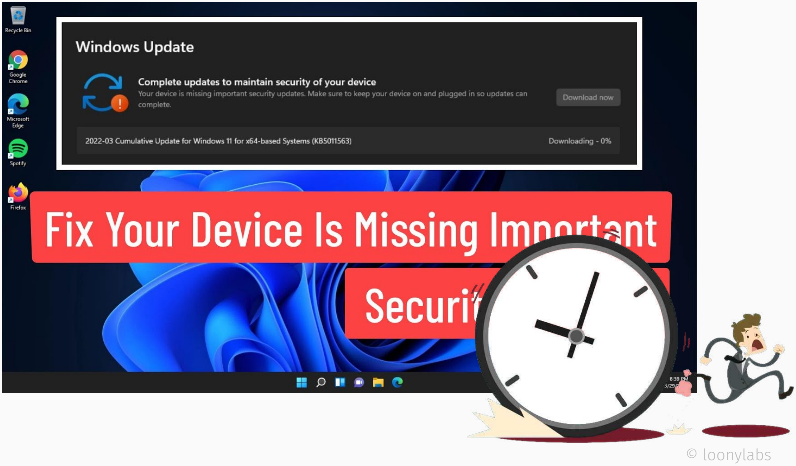
1. Introduction
2. Background
3. Forecast model
4. Conclusions

# Introduction

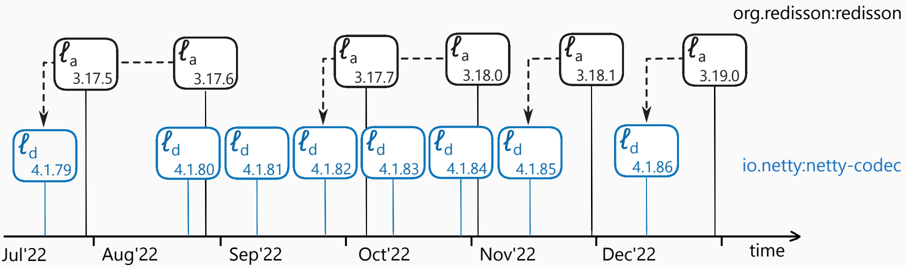
# Those annoying security updates



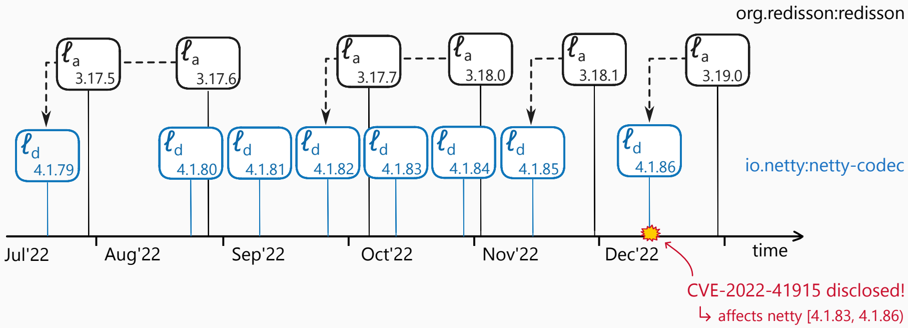
# Those annoying security updates



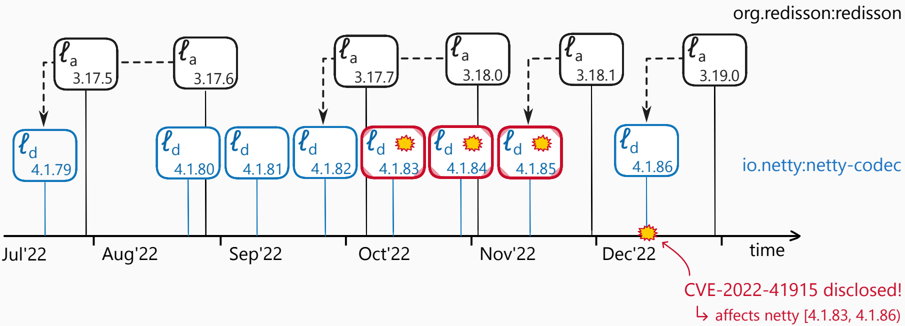
# Some motivation (plz!)



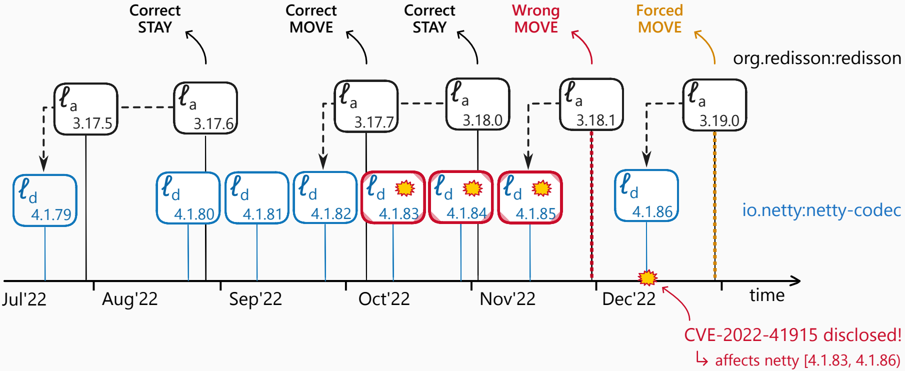
# Some motivation (plz!)



# Some motivation (plz!)



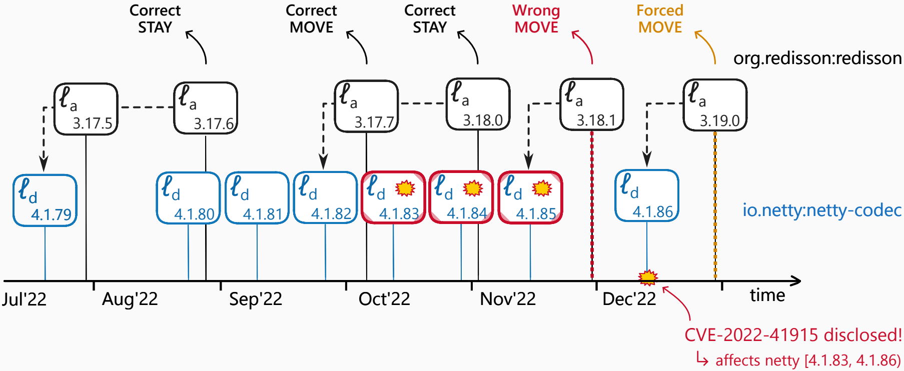
# Some motivation (plz!)



# Some motivation (plz!)

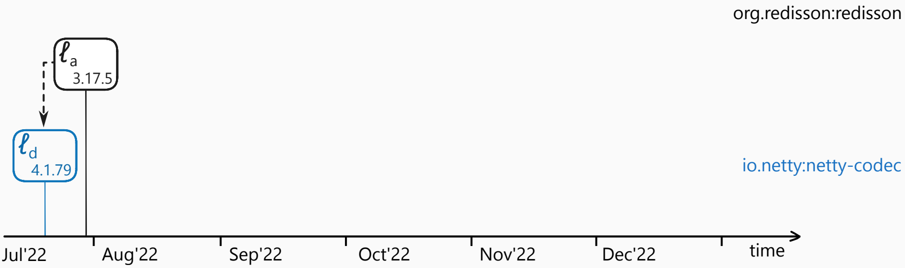
## Hindsight!

© j4p4n



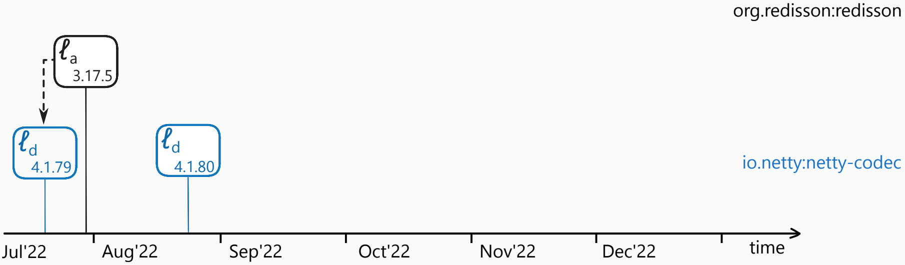
# Some motivation (plz!)

## Developer perspective in time:



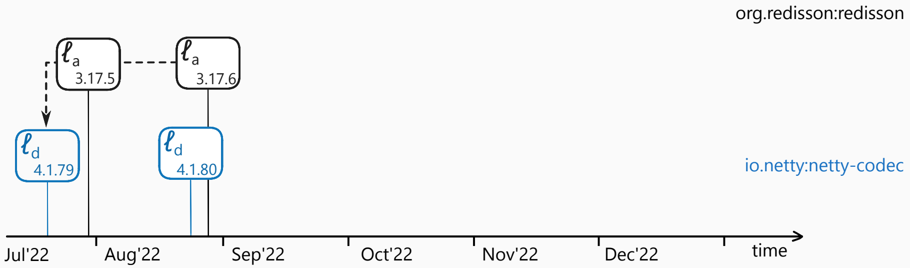
# Some motivation (plz!)

Developer perspective in time:



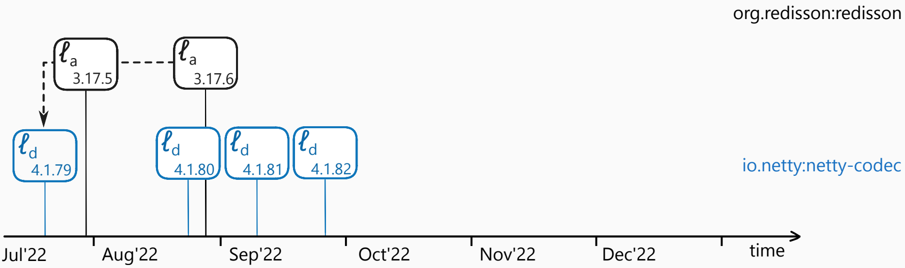
# Some motivation (plz!)

Developer perspective in time:



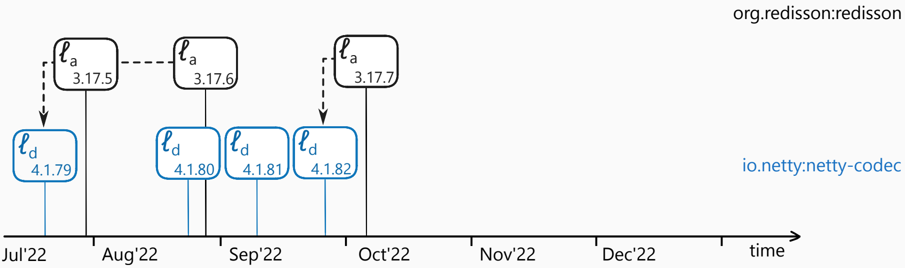
# Some motivation (plz!)

Developer perspective in time:



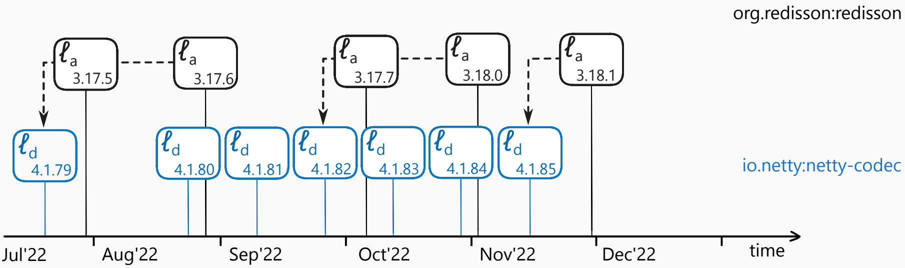
# Some motivation (plz!)

Developer perspective in time:

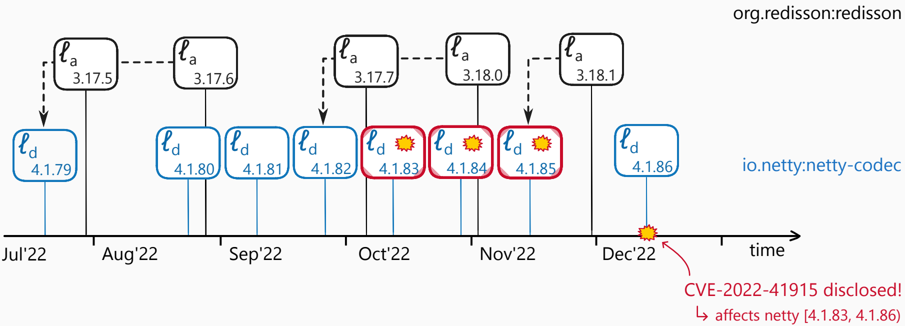


# Some motivation (plz!)

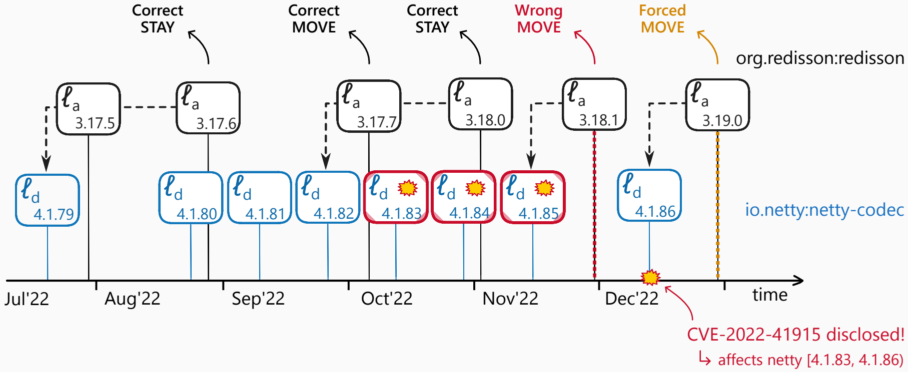
Developer perspective in time:



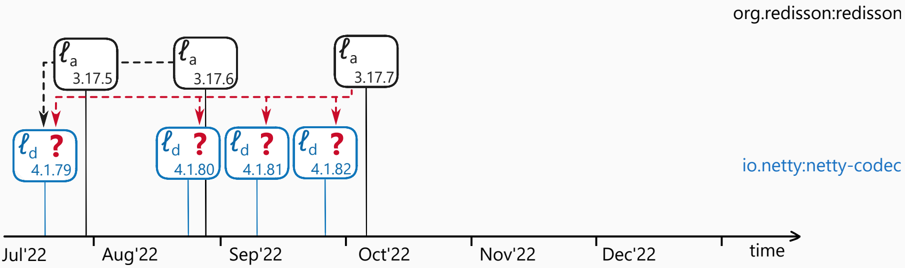
## Developer perspective in time:



## Developer perspective in time:



## Is there a best time to update?



# Questions

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

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

# 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 $\Pr(\text{vuln.})$?

- ▷ best time to update?

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

- ▷ measurable software metrics

# Outta scopes

1. Unpublished/Undetected vulnerabilities:
    - we study publication of CVEs;

# Outta scopes

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

# Outta scopes

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;

# Outta scopes

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!

# Background



| Work | Goal | | Data | | | | Method | | | Approach | | | Projects/Libs. | | Purport |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Disc. | Pred. | CVEs | Code | VCS | Dep. | Corr. | Clas. | T.Ser. | AH | SA | ML | Language | # | |
| [WTT+24] | ✓ | | | ✓ | | | ✓ | ✓ | | | | ✓ | C/C++ | 20 | Find vulnerabilities regardless of existent logs such as CVEs (although CWEs may be used). This includes formal methods and static/dynamic code analysis. |
| [BES+20] | ✓ | | | ✓ | | | | ✓ | | | | ✓ | C | 3 | |
| [AT17] | ✓ | | | | ✓ | | ✓ | ✓ | | | | ✓ | PHP | 3 | |
| [BCH+14] | ✓ | | | ✓ | ✓ | | | ✓ | | ✓ | | | C/C++, PHP, Java, JS, SQL | 10 | |
| [LYZ+23] | ✓ | | | ✓ | ✓ | | | ✓ | | ✓ | | ✓ | C, Java | 549 | Detect known vulnerabilities (and their correlation to developer activity metrics) from VCS only—e.g. commit churn, peer comments, etc. |
| [LKKL14] | ✓ | | ✓ | | ✓ | | | ✓ | | ✓ | | | C | 3 | |
| [MSM+13] | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | | | C | 1 | |
| [MW10] | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | C, ASM | 3 | |
| [CKDR21] | ✓ | | ✓ | ✓ | | | | ✓ | | | | ✓ | C/C++ | 3 | Detect known vulnerabilities (and their correlation to code metrics) from code only—e.g. number of classes, code cloning, cyclomatic complexity, etc. |
| [GOP21] | ✓ | | ✓ | ✓ | | | | ✓ | | | | ✓ | Java | 7 | |
| [SAC21] | ✓ | | ✓ | ✓ | | | ✓ | ✓ | | | ✓ | ✓ | Java | 4 | |
| [SDW17] | ✓ | | ✓ | ✓ | | | ✓ | | | | ✓ | | Java | 3 | |
| [SW17] | ✓ | | ✓ | ✓ | | | ✓ | | | | ✓ | | Java | 5 | |
| [SMM+12] | ✓ | | ✓ | ✓ | | | | ✓ | | ✓ | | | C | 7 | |
| [AL21] | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | | | | ✓ | C/C++ | >150k | 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. |
| [KWLO17] | ✓ | | ✓ | ✓ | ✓ | | | ✓ | | ✓ | | | C/C++ | 8 | |
| [AFA16] | ✓ | | ✓ | ✓ | ✓ | | ✓ | | | | ✓ | | C/C++ | 5 | |
| [CZ11] | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | | | ✓ | ✓ | C/C++, Java | 1 | |
| [SMWO11] | ✓ | | ✓ | ✓ | ✓ | | ✓ | | | | ✓ | ✓ | C/C++ | 2 | |
| [PPP+22] | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | | ✓ | | | Java | 500 | Detect known vulnerabilities using code or VCS, via dependency-aware models that can find the offending code, to aid in its solution (own vs. 3$^{rd}$ party lib). |
| [LCF+22] | ✓ | | ✓ | ✓ | | ✓ | | ✓ | | ✓ | | | JS | 624 | |
| [LST+21] | ✓ | | ✓ | ✓ | | ✓ | | ✓ | | | | ✓ | Java | >300k | |
| [PSS+21] | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | | Java, Ruby, Python | 450 | |
| [LRW22] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 4 | Time regression to predict vulnerabilities from NVD logs, but the models do not use domain-specific data relevant for security. |
| [YPWS20] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 9 | |
| [Las16] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 25 | |
| [RNR15] | | ✓ | ✓ | | | | | | ✓ | | ✓ | | Agnostic | 5 | |

| Work | Goal: Disc. | Goal: Pred. | Data: CVEs | Data: Code | Data: VCS | Data: Dep. | Method: Corr. | Method: Clas. | Method: T.Ser. | Approach: AH | Approach: SA | Approach: ML | Projects/Libs.: Language | Projects/Libs.: # | Purport |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| [WTT+24] | ✓ | | | ✓ | | | ✓ | ✓ | | | | ✓ | C/C++ | 20 | Find vulnerabilities regardless of existent logs such as CVEs (although CWEs may be used). This includes formal methods and static/dynamic code analysis. |
| [BES+20] | ✓ | | | ✓ | | | | ✓ | | | | ✓ | C | 3 | |
| [AT17] | ✓ | | | | ✓ | | ✓ | ✓ | | | | ✓ | PHP | 3 | |
| [BCH+14] | ✓ | | | ✓ | ✓ | | | ✓ | | ✓ | | | C/C++, PHP, Java, JS, SQL | 10 | |
| [LYZ+23] | ✓ | | | ✓ | ✓ | | | ✓ | | ✓ | | ✓ | C, Java | 549 | Detect known vulnerabilities (and their correlation to developer activity metrics) from VCS only—e.g. commit churn, peer comments, etc. |
| [LKKL14] | ✓ | | ✓ | | ✓ | | | ✓ | | ✓ | | | C | 3 | |
| [MSM+13] | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | | | C | 1 | |
| [MW10] | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | C, ASM | 3 | |
| [CKDR21] | ✓ | | ✓ | ✓ | | | | ✓ | | | | ✓ | C/C++ | 3 | Detect known vulnerabilities (and their correlation to code metrics) from code only—e.g. number of classes, code cloning, cyclomatic complexity, etc. |
| [GOP21] | ✓ | | ✓ | ✓ | | | | ✓ | | | | ✓ | Java | 7 | |
| [SAC21] | ✓ | | ✓ | ✓ | | | ✓ | ✓ | | | ✓ | ✓ | Java | 4 | |
| [SDW17] | ✓ | | ✓ | ✓ | | | ✓ | | | | ✓ | | Java | 3 | |
| [SW17] | ✓ | | ✓ | ✓ | | | ✓ | | | | ✓ | | Java | 5 | |
| [SMM+12] | ✓ | | ✓ | ✓ | | | | ✓ | | ✓ | | | C | 7 | |
| [AL21] | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | | | | ✓ | C/C++ | >150k | 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. |
| [KWLO17] | ✓ | | ✓ | ✓ | ✓ | | | ✓ | | ✓ | | | C/C++ | 8 | |
| [AFA16] | ✓ | | ✓ | ✓ | ✓ | | ✓ | | | | ✓ | | C/C++ | 5 | |
| [CZ11] | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | | | ✓ | ✓ | C/C++, Java | 1 | |
| [SMWO11] | ✓ | | ✓ | ✓ | ✓ | | ✓ | | | | ✓ | ✓ | C/C++ | 2 | |
| [PPP+22] | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | | ✓ | | | Java | 500 | Detect known vulnerabilities using code or VCS, via dependency-aware models that can find the offending code, to aid in its solution (own vs. 3rd party lib). |
| [LCF+22] | ✓ | | ✓ | ✓ | | ✓ | | ✓ | | ✓ | | | JS | 624 | |
| [LST+21] | ✓ | | ✓ | ✓ | | ✓ | | ✓ | | | | ✓ | Java | >300k | |
| [PSS+21] | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | | Java, Ruby, Python | 450 | |
| [LRW22] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 4 | Time regression to predict vulnerabilities from NVD logs, but the models do not use domain-specific data relevant for security. |
| [YPWS20] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 9 | |
| [Las16] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 25 | |
| [RNR15] | | ✓ | ✓ | | | | | | ✓ | | ✓ | | Agnostic | 5 | |

Most works try to discover current vulnerabilities, not predict future ones

| Work | Goal: Disc. | Goal: Pred. | Data: CVEs | Data: Code | Data: VCS | Data: Dep. | Method: Corr. | Method: Clas. | Method: T.Ser. | Approach: AH | Approach: SA | Approach: ML | Projects/Libs.: Language | Projects/Libs.: # | Purport |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| [WTT+24] | ✓ | | | ✓ | | | ✓ | ✓ | | | | ✓ | C/C++ | 20 | Find vulnerabilities regardless of existent logs such as CVEs (although CWEs may be used). This includes formal methods and static/dynamic code analysis. |
| [BES+20] | ✓ | | | ✓ | | | | ✓ | | | | ✓ | C | 3 | |
| [AT17] | ✓ | | | | ✓ | | ✓ | ✓ | | | | ✓ | PHP | 3 | |
| [BCH+14] | ✓ | | | ✓ | ✓ | | | ✓ | | ✓ | | | C/C++, PHP, Java, JS, SQL | 10 | |
| [LYZ+23] | ✓ | | | ✓ | ✓ | | | ✓ | | ✓ | | ✓ | C, Java | 549 | Detect known vulnerabilities (and their correlation to developer activity metrics) from VCS only—e.g. commit churn, peer comments, etc. |
| [LKKL14] | ✓ | | ✓ | | ✓ | | | ✓ | | ✓ | | | C | 3 | |
| [MSM+13] | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | | | C | 1 | |
| [MW10] | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | C, ASM | 3 | |
| [CKDR21] | ✓ | | ✓ | ✓ | | | | ✓ | | | | ✓ | C/C++ | 3 | Detect known vulnerabilities (and their correlation to code metrics) from code only—e.g. number of classes, code cloning, cyclomatic complexity, etc. |
| [GOP21] | ✓ | | ✓ | ✓ | | | | ✓ | | | | ✓ | Java | 7 | |
| [SAC21] | ✓ | | ✓ | ✓ | | | ✓ | ✓ | | | ✓ | ✓ | Java | 4 | |
| [SDW17] | ✓ | | ✓ | ✓ | | | ✓ | | | | ✓ | | Java | 3 | |
| [SW17] | ✓ | | ✓ | ✓ | | | ✓ | | | | ✓ | | Java | 5 | |
| [SMM+12] | ✓ | | ✓ | ✓ | | | | ✓ | | ✓ | | | C | 7 | |
| [AL21] | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | | | | ✓ | C/C++ | >150k | 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. |
| [KWLO17] | ✓ | | ✓ | ✓ | ✓ | | | ✓ | | ✓ | | | C/C++ | 8 | |
| [AFA16] | ✓ | | ✓ | ✓ | ✓ | | ✓ | | | | ✓ | | C/C++ | 5 | |
| [CZ11] | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | | | ✓ | ✓ | C/C++, Java | 1 | |
| [SMWO11] | ✓ | | ✓ | ✓ | ✓ | | ✓ | | | | ✓ | ✓ | C/C++ | 2 | |
| [PPP+22] | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | | ✓ | | | Java | 500 | Detect known vulnerabilities using code or VCS, via dependency-aware models that can find the offending code, to aid in its solution (own vs. 3rd party lib). |
| [LCF+22] | ✓ | | ✓ | ✓ | | ✓ | | ✓ | | ✓ | | | JS | 624 | |
| [LST+21] | ✓ | | ✓ | ✓ | | ✓ | | ✓ | | | | ✓ | Java | >300k | |
| [PSS+21] | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | | Java, Ruby, Python | 450 | |
| [LRW22] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 4 | Time regression to predict vulnerabilities from NVD logs, but the models do not use domain-specific data relevant for security. |
| [YPWS20] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 9 | |
| [Las16] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 25 | |
| [RNR15] | | ✓ | ✓ | | | | | | ✓ | | ✓ | | Agnostic | 5 | |

Most works try to discover current vulnerabilities, not predict future ones

Most works disregard the code dependency tree

# State of the *ART* — Models to predict vulnerabilities

| Work | Goal: Disc. | Goal: Pred. | Data: CVEs | Data: Code | Data: VCS | Data: Dep. | Method: Corr. | Method: Clas. | Method: T.Ser. | Approach: AH | Approach: SA | Approach: ML | Projects/Libs.: Language | Projects/Libs.: # | Purport |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| [WTT+24] | ✓ | | | ✓ | | | ✓ | ✓ | | | | ✓ | C/C++ | 20 | Find vulnerabilities regardless of existent logs such as CVEs (although CWEs may be used). This includes formal methods and static/dynamic code analysis. |
| [BES+20] | ✓ | | | ✓ | | | | ✓ | | | | ✓ | C | 3 | |
| [AT17] | ✓ | | | | ✓ | | ✓ | ✓ | | | | ✓ | PHP | 3 | |
| [BCH+14] | ✓ | | | ✓ | ✓ | | | ✓ | | ✓ | | | C/C++, PHP, Java, JS, SQL | 10 | |
| [LYZ+23] | ✓ | | | ✓ | ✓ | | | ✓ | | ✓ | | ✓ | C, Java | 549 | Detect known vulnerabilities (and their correlation to developer activity metrics) from VCS only—e.g. commit churn, peer comments, etc. |
| [LKKL14] | ✓ | | ✓ | | ✓ | | | ✓ | | ✓ | | | C | 3 | |
| [MSM+13] | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | | | C | 1 | |
| [MW10] | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | C, ASM | 3 | |
| [CKDR21] | ✓ | | ✓ | ✓ | | | | ✓ | | | | ✓ | C/C++ | 3 | Detect known vulnerabilities (and their correlation to code metrics) from code only—e.g. number of classes, code cloning, cyclomatic complexity, etc. |
| [GOP21] | ✓ | | ✓ | ✓ | | | | ✓ | | | | ✓ | Java | 7 | |
| [SAC21] | ✓ | | ✓ | ✓ | | | ✓ | ✓ | | | ✓ | ✓ | Java | 4 | |
| [SDW17] | ✓ | | ✓ | ✓ | | | ✓ | | | | ✓ | | Java | 3 | |
| [SW17] | ✓ | | ✓ | ✓ | | | ✓ | | | | ✓ | | Java | 5 | |
| [SMM+12] | ✓ | | ✓ | ✓ | | | | ✓ | | ✓ | | | C | 7 | |
| [AL21] | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | | | | ✓ | C/C++ | >150k | 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. |
| [KWLO17] | ✓ | | ✓ | ✓ | ✓ | | | ✓ | | ✓ | | | C/C++ | 8 | |
| [AFA16] | ✓ | | ✓ | ✓ | ✓ | | ✓ | | | | ✓ | | C/C++ | 5 | |
| [CZ11] | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | | | ✓ | ✓ | C/C++, Java | 1 | |
| [SMWO11] | ✓ | | ✓ | ✓ | ✓ | | ✓ | | | | ✓ | ✓ | C/C++ | 2 | |
| [PPP+22] | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | | ✓ | | | Java | 500 | Detect known vulnerabilities using code or VCS, via dependency-aware models that can find the offending code, to aid in its solution (own vs. 3rd party lib). |
| [LCF+22] | ✓ | | ✓ | ✓ | | ✓ | | ✓ | | ✓ | | | JS | 624 | |
| [LST+21] | ✓ | | ✓ | ✓ | | ✓ | | ✓ | | | | ✓ | Java | >300k | |
| [PSS+21] | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | | Java, Ruby, Python | 450 | |
| [LRW22] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 4 | Time regression to predict vulnerabilities from NVD logs, but the models do not use domain-specific data relevant for security. |
| [YPWS20] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 9 | |
| [Las16] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 25 | |
| [RNR15] | | ✓ | ✓ | | | | | | ✓ | | ✓ | | Agnostic | 5 | |

Most works try to discover current vulnerabilities, not predict future ones

Most works disregard the code dependency tree

Most works do not consider time in their analyses

| Work | Goal: Disc. | Goal: Pred. | Data: CVEs | Data: Code | Data: VCS | Data: Dep. | Method: Corr. | Method: Clas. | Method: T.Ser. | Approach: AH | Approach: SA | Approach: ML | Projects/Libs.: Language | Projects/Libs.: # | Purport |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| [WTT+24] | ✓ | | | ✓ | | | ✓ | ✓ | | | | ✓ | C/C++ | 20 | Find vulnerabilities regardless of existent logs such as CVEs (although CWEs may be used). This includes formal methods and static/dynamic code analysis. |
| [BES+20] | ✓ | | | ✓ | | | | ✓ | | | | ✓ | C | 3 | |
| [AT17] | ✓ | | | | ✓ | | ✓ | ✓ | | | | ✓ | PHP | 3 | |
| [BCH+14] | ✓ | | | ✓ | ✓ | | | ✓ | | ✓ | | | C/C++, PHP, Java, JS, SQL | 10 | |
| [LYZ+23] | ✓ | | | ✓ | ✓ | | | ✓ | | ✓ | | ✓ | C, Java | 549 | Detect known vulnerabilities (and their correlation to developer activity metrics) from VCS only—e.g. commit churn, peer comments, etc. |
| [LKKL14] | ✓ | | ✓ | | ✓ | | | ✓ | | ✓ | | | C | 3 | |
| [MSM+13] | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | | | C | 1 | |
| [MW10] | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | C, ASM | 3 | |
| [CKDR21] | ✓ | | ✓ | ✓ | | | | ✓ | | | | ✓ | C/C++ | 3 | Detect known vulnerabilities (and their correlation to code metrics) from code only—e.g. number of classes, code cloning, cyclomatic complexity, etc. |
| [GOP21] | ✓ | | ✓ | ✓ | | | | ✓ | | | | ✓ | Java | 7 | |
| [SAC21] | ✓ | | ✓ | ✓ | | | ✓ | ✓ | | | ✓ | ✓ | Java | 4 | |
| [SDW17] | ✓ | | ✓ | ✓ | | | ✓ | | | | ✓ | | Java | 3 | |
| [SW17] | ✓ | | ✓ | ✓ | | | ✓ | | | | ✓ | | Java | 5 | |
| [SMM+12] | ✓ | | ✓ | ✓ | | | | ✓ | | ✓ | | | C | 7 | |
| [AL21] | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | | | | ✓ | C/C++ | >150k | 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. |
| [KWLO17] | ✓ | | ✓ | ✓ | ✓ | | | ✓ | | ✓ | | | C/C++ | 8 | |
| [AFA16] | ✓ | | ✓ | ✓ | ✓ | | ✓ | | | | ✓ | | C/C++ | 5 | |
| [CZ11] | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | | | ✓ | ✓ | C/C++, Java | 1 | |
| [SMWO11] | ✓ | | ✓ | ✓ | ✓ | | ✓ | | | | ✓ | ✓ | C/C++ | 2 | |
| [PPP+22] | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | | ✓ | | | Java | 500 | Detect known vulnerabilities using code or VCS, via dependency-aware models that can find the offending code, to aid in its solution (own vs. 3rd party lib). |
| [LCF+22] | ✓ | | ✓ | ✓ | | ✓ | | ✓ | | ✓ | | | JS | 624 | |
| [LST+21] | ✓ | | ✓ | ✓ | | ✓ | | ✓ | | | | ✓ | Java | >300k | |
| [PSS+21] | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | | Java, Ruby, Python | 450 | |
| [LRW22] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 4 | Time regression to predict vulnerabilities from NVD logs, but the models do not use domain-specific data relevant for security. |
| [YPWS20] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 9 | |
| [Las16] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 25 | |
| [RNR15] | | ✓ | ✓ | | | | | | ✓ | | ✓ | | Agnostic | 5 | |

Most works try to discover current vulnerabilities, not predict future ones

Most works disregard the code dependency tree

Most works do not consider time in their analyses

Disregarded security data

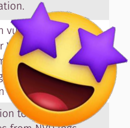
| Work | Goal: Disc. | Goal: Pred. | Data: CVEs | Data: Code | Data: VCS | Data: Dep. | Method: Corr. | Method: Clas. | Method: T.Ser. | Approach: AH | Approach: SA | Approach: ML | Projects/Libs.: Language | Projects/Libs.: # | Purport |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| [WTT+24] | ✓ | | | ✓ | | | ✓ | ✓ | | | | ✓ | C/C++ | 20 | Find vulnerabilities regardless of existent logs such as CVEs (although CWEs may be used). This includes formal methods and static/dynamic code analysis. |
| [BES+20] | ✓ | | | ✓ | | | | ✓ | | | | ✓ | C | 3 | |
| [AT17] | ✓ | | | | ✓ | | ✓ | ✓ | | | | ✓ | PHP | 3 | |
| [BCH+14] | ✓ | | | ✓ | ✓ | | | ✓ | | ✓ | | | C/C++, PHP, Java, JS, SQL | 10 | |
| [LYZ+23] | ✓ | | | ✓ | ✓ | | | ✓ | | ✓ | | ✓ | C, Java | 549 | Detect known vulnerabilities (and their correlation to developer activity metrics) from VCS only—e.g. commit churn, peer comments, etc. |
| [LKKL14] | ✓ | | ✓ | | ✓ | | | ✓ | | ✓ | | | C | 3 | |
| [MSM+13] | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | | | C | 1 | |
| [MW10] | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | C, ASM | 3 | |
| [CKDR21] | ✓ | | ✓ | ✓ | | | | ✓ | | | | ✓ | C/C++ | 3 | Detect known vulnerabilities (and their correlation to code metrics) from code only—e.g. number of classes, code cloning, cyclomatic complexity, etc. |
| [GOP21] | ✓ | | ✓ | ✓ | | | | ✓ | | | | ✓ | Java | 7 | |
| [SAC21] | ✓ | | ✓ | ✓ | | | ✓ | ✓ | | | ✓ | ✓ | Java | 4 | |
| [SDW17] | ✓ | | ✓ | ✓ | | | ✓ | | | | ✓ | | Java | 3 | |
| [SW17] | ✓ | | ✓ | ✓ | | | ✓ | | | | ✓ | | Java | 5 | |
| [SMM+12] | ✓ | | ✓ | ✓ | | | | ✓ | | ✓ | | | C | 7 | |
| [AL21] | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | | | | ✓ | C/C++ | >150k | 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. |
| [KWLO17] | ✓ | | ✓ | ✓ | ✓ | | | ✓ | | ✓ | | | C/C++ | 8 | |
| [AFA16] | ✓ | | ✓ | ✓ | ✓ | | ✓ | | | | ✓ | | C/C++ | 5 | |
| [CZ11] | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | | | ✓ | ✓ | C/C++, Java | 1 | |
| [SMWO11] | ✓ | | ✓ | ✓ | ✓ | | ✓ | | | | ✓ | ✓ | C/C++ | 2 | |
| [PPP+22] | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | | ✓ | | | Java | 500 | Detect known vu[...] using code or [...] ency-aware m[...] the offending [...] solution (own |
| [LCF+22] | ✓ | | ✓ | ✓ | | ✓ | | ✓ | | ✓ | | | JS | 624 | |
| [LST+21] | ✓ | | ✓ | ✓ | | ✓ | | ✓ | | | | ✓ | Java | >300k | |
| [PSS+21] | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | | Java, Ruby, Python | 450 | |
| [LRW22] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 4 | Time regression to [...] vulnerabilities from NVD logs, but the models do not use domain-specific data relevant for security. |
| [YPWS20] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 9 | |
| [Las16] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 25 | |
| [RNR15] | | ✓ | ✓ | | | | | | ✓ | | ✓ | | Agnostic | 5 | |

Most works try to discover current vulnerabilities, not predict future ones

Most works disregard the code dependency tree

Most works do not consider time in their analyses

Disregarded security data



**Q2** Pr(vuln.) as function of software metrics

**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

**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...)

**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

**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)

# Gap analysis

- 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).

# 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.

# 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.

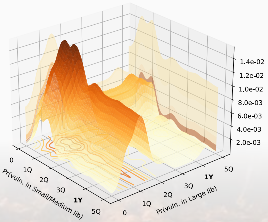
**We propose white-box model(s) to fill these gaps**

# Forecast model

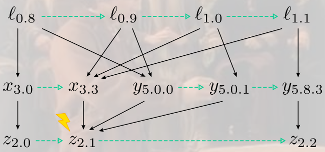


# Forecast model

CVE root-lib PDFs

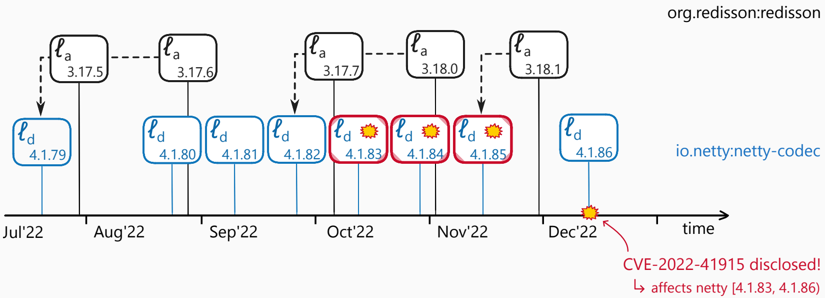




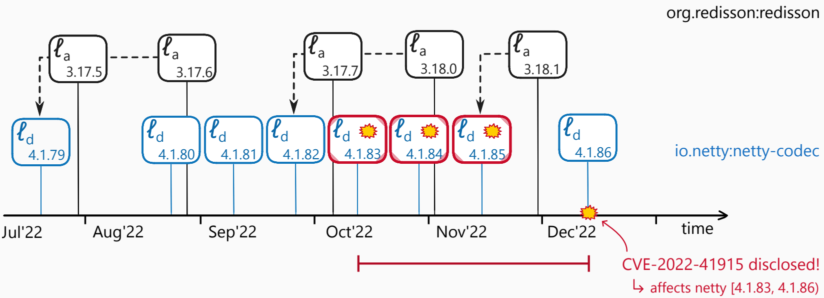




Time Dependency Trees

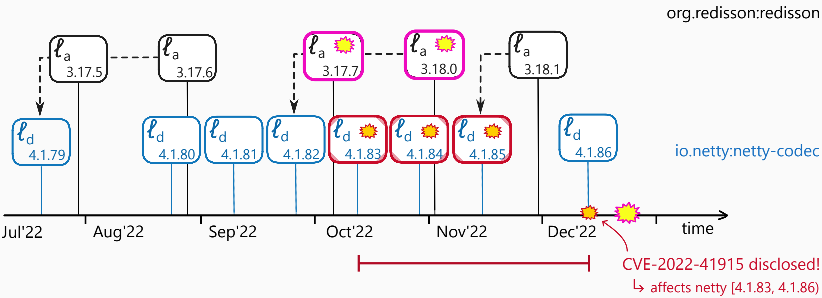
# Publication of CVE since time of code release



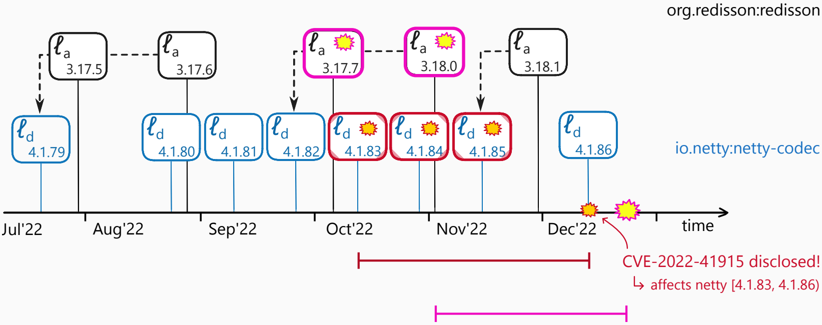
# Publication of CVE since time of code release



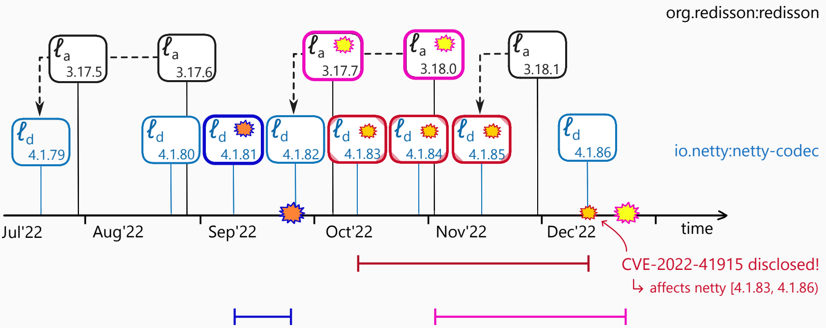
# Publication of CVE since time of code release



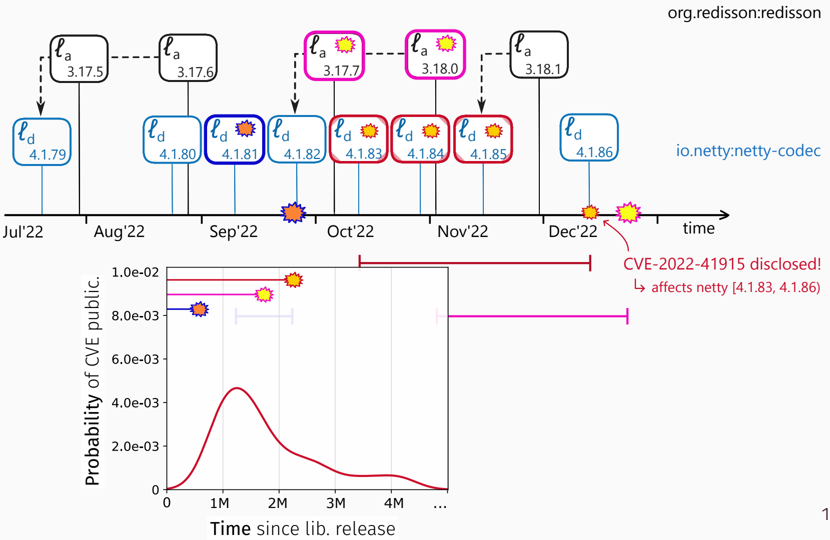
# Publication of CVE since time of code release



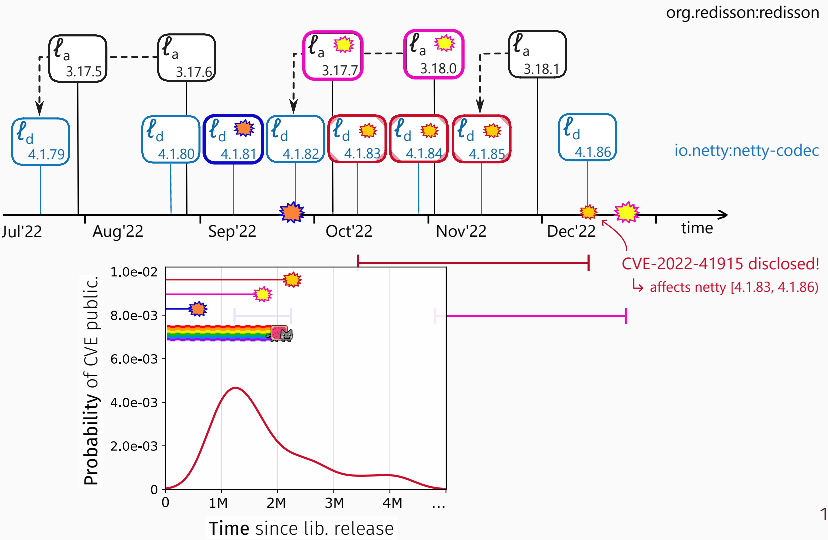
# Publication of CVE since time of code release



# Publication of CVE since time of code release



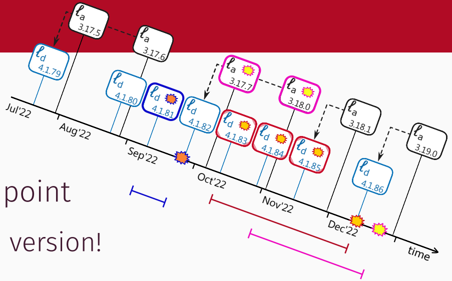
# Publication of CVE since time of code release



# Rules of the game

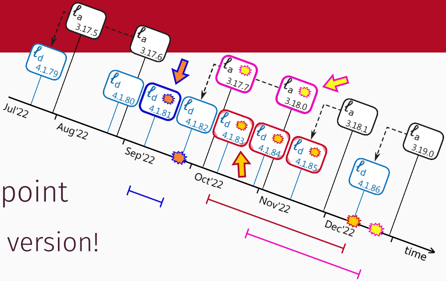
- Count each CVE as one data point
  - must choose one affected version!

# 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!

# 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!)

    🍏 🍊

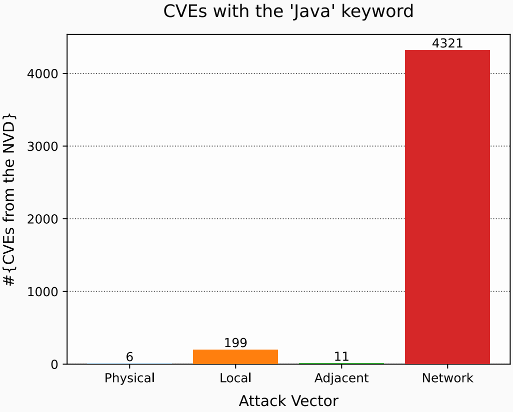
# Rules of the game

- 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

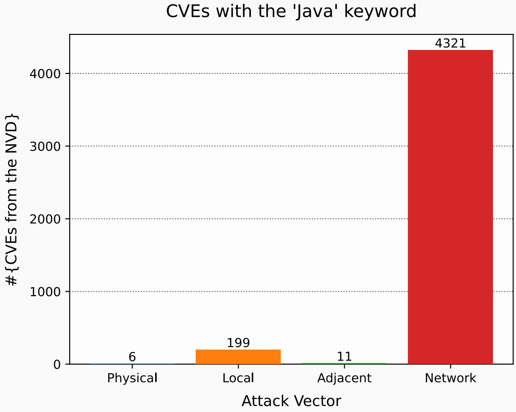
# Rules of the game

- 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 🤌

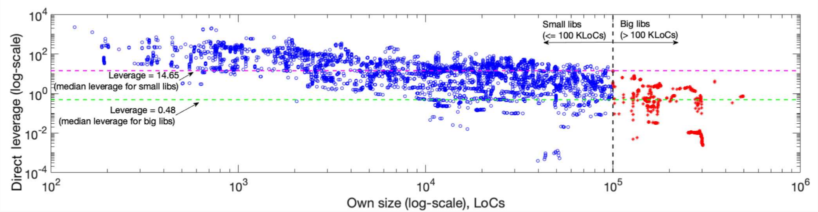
# Security-relevant code metrics

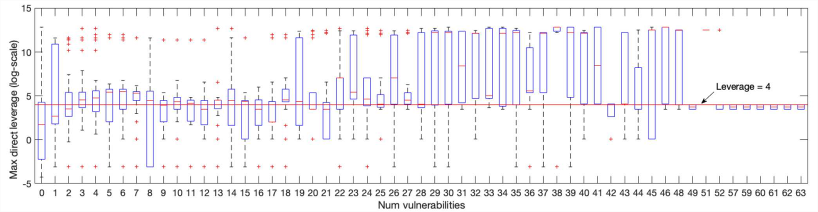


## Used in remote networks



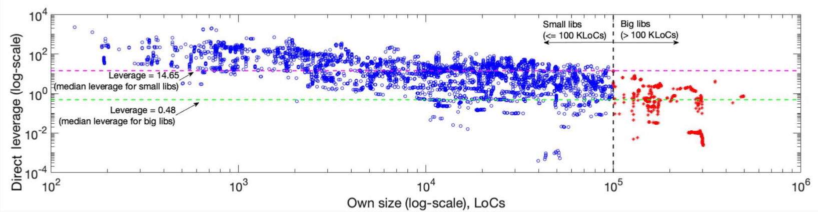
# Security-relevant code metrics

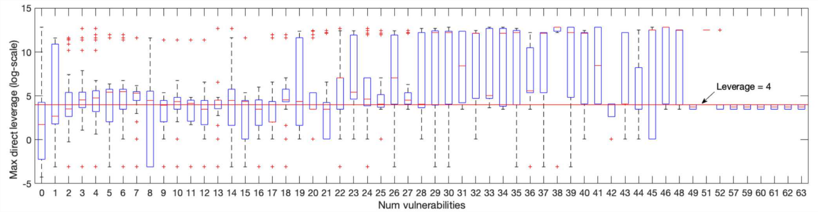


# Security-relevant code metrics

## (Own) Code size



# Security-relevant code metrics

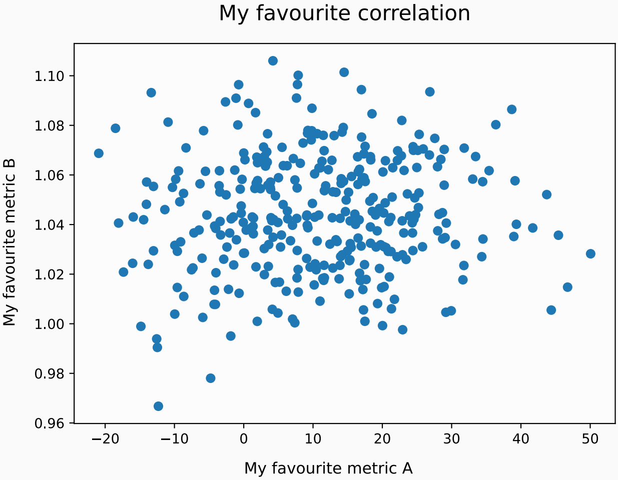
| Work | Goal: Disc. | Goal: Pred. | Data: CVEs | Data: Code | Data: VCS | Data: Dep. | Method: Corr. | Method: Clas. | Method: T.Ser. | Approach: AH | Approach: SA | Approach: ML | Projects/Libs.: Language | Projects/Libs.: # | Purport |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| [WTT$^{+}$24] | ✓ | | | ✓ | | | ✓ | ✓ | | | | ✓ | C/C++ | 20 | Find vulnerabilities regardless of existent logs such as CVEs (although CWEs may be used). This includes formal methods and static/dynamic code analysis. |
| [BES$^{+}$20] | ✓ | | | ✓ | | | | ✓ | | | | ✓ | C | 3 | |
| [AT17] | ✓ | | | | ✓ | | ✓ | ✓ | | | | ✓ | PHP | 3 | |
| [BCH$^{+}$14] | ✓ | | | ✓ | ✓ | | | ✓ | | ✓ | | | C/C++, PHP, Java, JS, SQL | 10 | |
| [LYZ$^{+}$23] | ✓ | | | ✓ | ✓ | | | ✓ | | ✓ | | ✓ | C, Java | 549 | Detect known vulnerabilities (and their correlation to developer activity metrics) from VCS only—e.g. commit churn, peer comments, etc. |
| [LKKL14] | ✓ | | ✓ | | ✓ | | | ✓ | | ✓ | | | C | 3 | |
| [MSM$^{+}$13] | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | | | C | 1 | |
| [MW10] | ✓ | | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | | C, ASM | 3 | |
| [CKDR21] | ✓ | | ✓ | ✓ | | | | ✓ | | | | ✓ | C/C++ | 3 | Detect known vulnerabilities (and their correlation to code metrics) from code only—e.g. number of classes, code cloning, cyclomatic complexity, etc. |
| [GOP21] | ✓ | | ✓ | ✓ | | | | ✓ | | | | ✓ | Java | 7 | |
| [SAC21] | ✓ | | ✓ | ✓ | | | ✓ | ✓ | | | ✓ | ✓ | Java | 4 | |
| [SDW17] | ✓ | | ✓ | ✓ | | | ✓ | | | | ✓ | | Java | 3 | |
| [SW17] | ✓ | | ✓ | ✓ | | | ✓ | | | | ✓ | | Java | 5 | |
| [SMM$^{+}$12] | ✓ | | ✓ | ✓ | | | | ✓ | | ✓ | | | C | 7 | |
| [AL21] | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | | | | ✓ | C/C++ | >150k | Detect known vulnerabilities (and their corr. to code and developer activity metrics) from both code and VCS, but without considering the effect of dependencies in their propagation. |
| [KWLO17] | ✓ | | ✓ | ✓ | ✓ | | | ✓ | | ✓ | | | C/C++ | 8 | |
| [AFA16] | ✓ | | ✓ | ✓ | ✓ | | ✓ | | | | ✓ | | C/C++ | 5 | |
| [CZ11] | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | | | ✓ | ✓ | C/C++, Java | 1 | |
| [SMWO11] | ✓ | | ✓ | ✓ | ✓ | | ✓ | | | | ✓ | ✓ | C/C++ | 2 | |
| [PPP$^{+}$22] | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | | ✓ | | | Java | 500 | Detect known vulnerabilities using code or VCS, via dependency-aware models that can find the offending code, to aid in its solution (own vs. $3^{rd}$ party lib). |
| [LCF$^{+}$22] | ✓ | | ✓ | ✓ | | ✓ | | ✓ | | ✓ | | | JS | 624 | |
| [LST$^{+}$21] | ✓ | | ✓ | ✓ | | ✓ | | ✓ | | | | ✓ | Java | >300k | |
| [PSS$^{+}$21] | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | | Java, Ruby, Python | 450 | |
| [LRW22] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 4 | Time regression to predict vulnerabilities from NVD logs, but the models do not use domain-specific data relevant for security. |
| [YPWS20] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 9 | |
| [Las16] | | ✓ | ✓ | | | | | | ✓ | | ✓ | ✓ | Agnostic | 25 | |
| [RNR15] | | ✓ | ✓ | | | | | | ✓ | | ✓ | | Agnostic | 5 | |

# Security-relevant code metrics

Used in remote networks

Own-code size

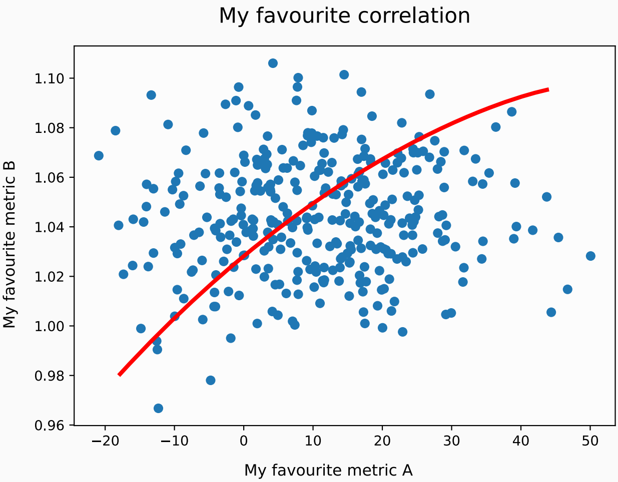
# Security-relevant code metrics




Used in remote networks

Own-code size

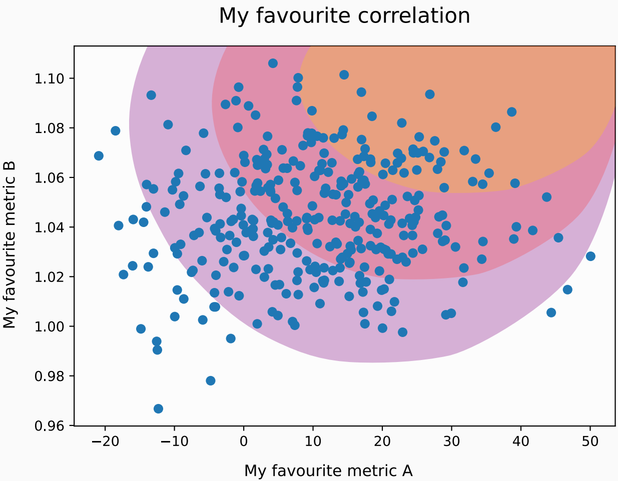
# Security-relevant code metrics




Used in remote networks

Own-code size

# Security-relevant code metrics

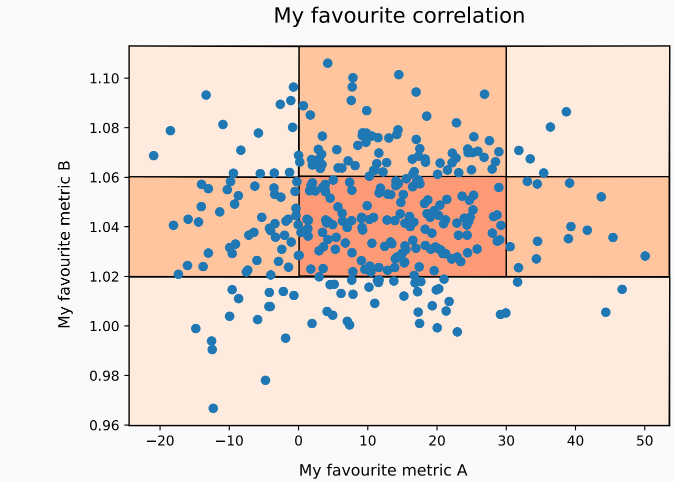



Used in remote networks

Own-code size

# Security-relevant code metrics

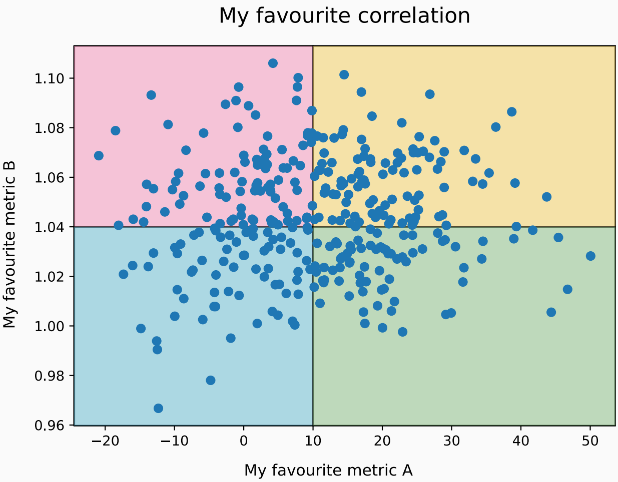
Used in remote networks




Own-code size

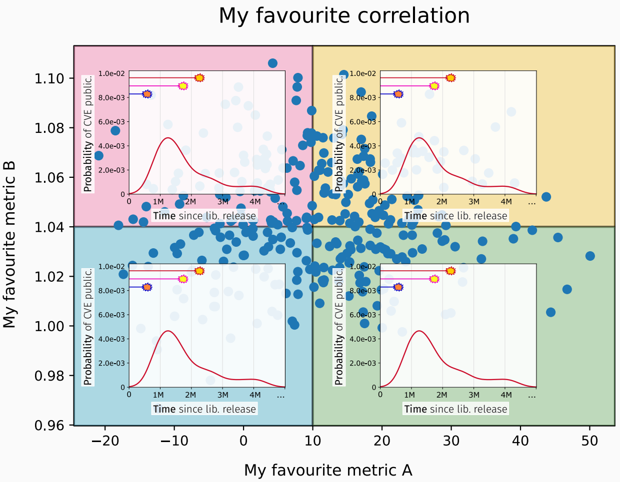
# Security-relevant code metrics

Used in remote networks




Own-code size

# Security-relevant code metrics

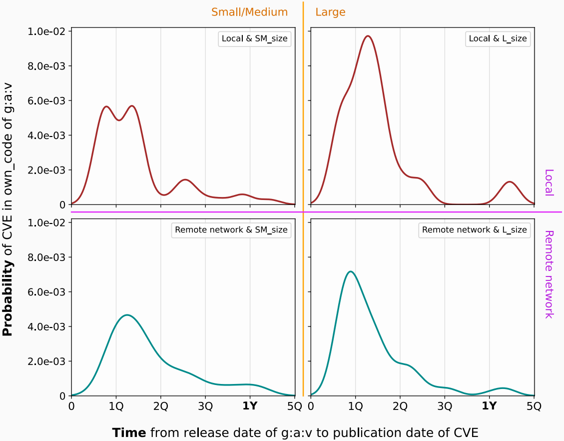



Used in remote networks

Own-code size

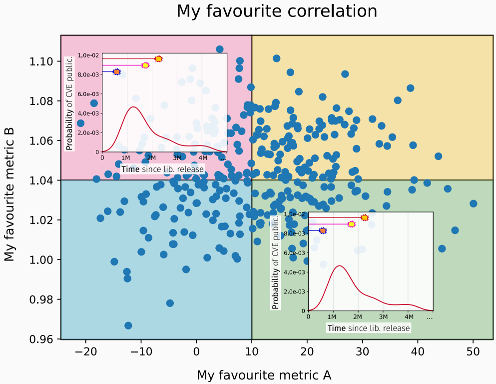
# Security-relevant code metrics

Used in remote networks

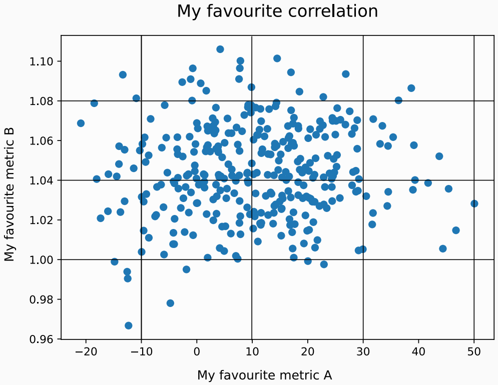



Own-code size

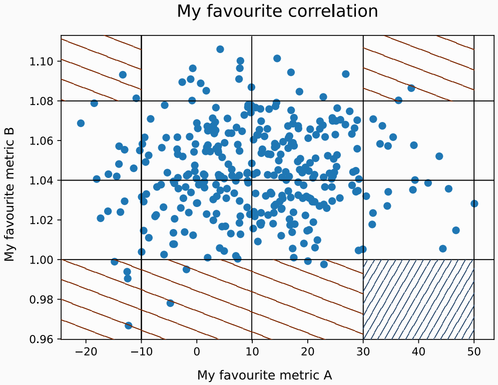
# On overfitting and rare events



# On overfitting and rare events



# 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

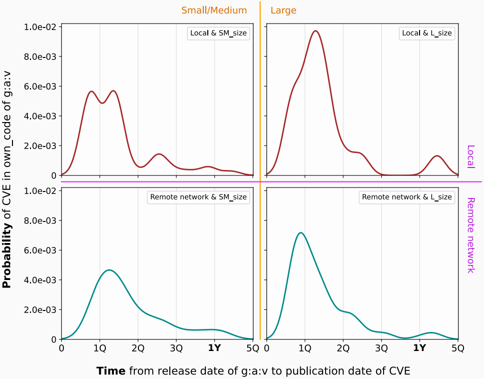
# On overfitting and rare events

- 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!**

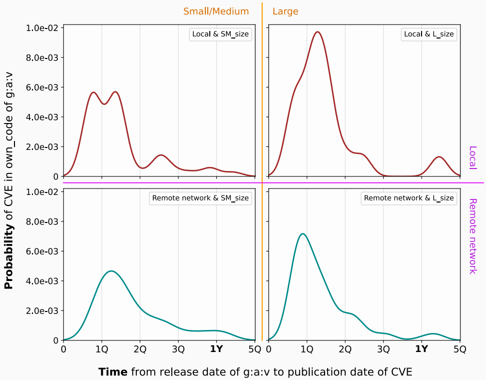
Gimme results

# Here ya go



Small/Medium
Large
Local & SM_size
Local & L_size
Remote network & SM_size
Remote network & L_size
Local
Remote network
Probability of CVE in own_code of g:a:v
Time from release date of g:a:v to publication date of CVE
1.0e-02
8.0e-03
6.0e-03
4.0e-03
2.0e-03
0
0
1Q
2Q
3Q
1Y
5Q

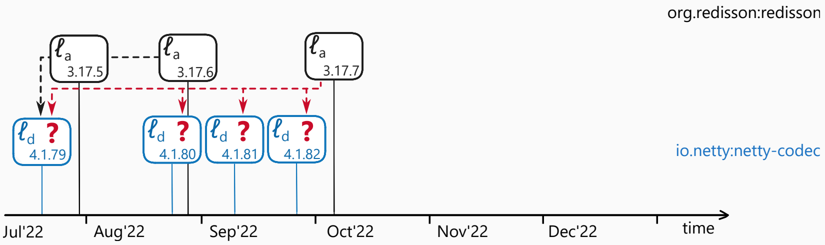
# Here ya go




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

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

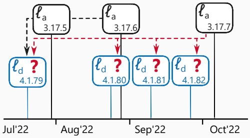
# Survival analysis on library update



# 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").

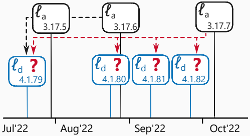
- $\ell_A$ was released on $t_A < t_0$, $\ell_B$ on $t_B < t_0$, $t_A \bowtie t_B$



# 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 $t_B < t_0$, $t_A \bowtie t_B$

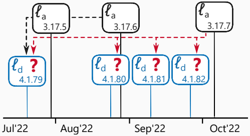



**Q:** $\mathrm{Pr}_{A,B}(t) =$ probability of vuln. of $A \xrightarrow{t} B$ as a function of $t$

# 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").

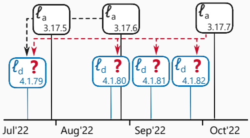
- ▷ $\ell_A$ was released on $t_A < t_0$, $\ell_B$ on $t_B < t_0$, $t_A \bowtie t_B$




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

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

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

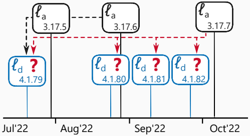
**A:** $\mathsf{Pr}_{A,B}(t) = 1 - \mathsf{SF}_A(t + \Delta t_A)\, \mathsf{CDF}_B(t + \Delta t_B)$ where $\Delta t_x \doteq |t_x - t_0|$

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").

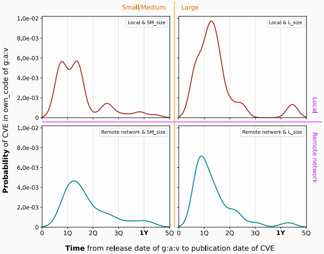
- $\ell_A$ was released on $t_A < t_0$, $\ell_B$ on $t_B < t_0$, $t_A \bowtie t_B$




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

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

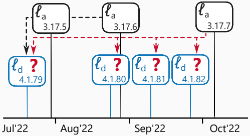
vuln. in $\ell_A$ before change    vuln. in $\ell_B$ after change



# Survival analysis on library update

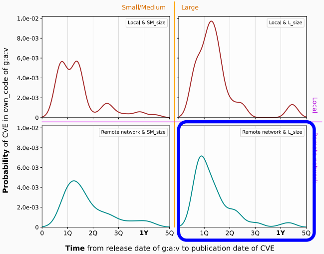
$A \xrightarrow{t} B$ means that we change from dependency $\ell_A$ to $\ell_B$ in $t$ time units counting from $t_0$ ("today").

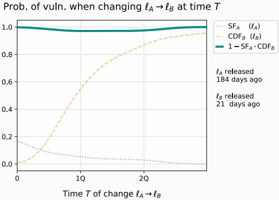
- $\ell_A$ was released on $t_A < t_0$, $\ell_B$ on $t_B < t_0$, $t_A \bowtie t_B$




**Q:** $\text{Pr}_{A,B}(t) =$ probability of vuln. of $A \xrightarrow{t} B$ 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 $\Delta t_x \doteq |t_x - t_0|$



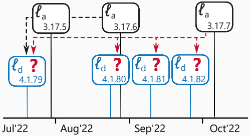




$t_A = 184$ days
$t_B = 21$ days

# Survival analysis on library update

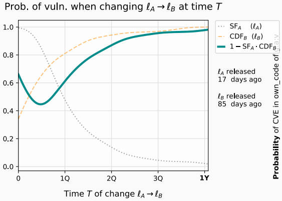
$A \xrightarrow{t} B$ means that we change from dependency $\ell_A$ to $\ell_B$ in $t$ time units counting from $t_0$ ("today").

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

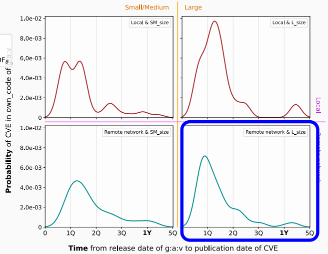


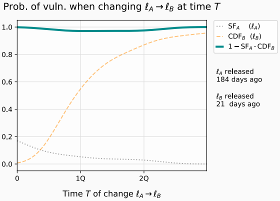

**Q:** $\mathsf{Pr}_{A,B}(t)$ = probability of vuln. of $A \xrightarrow{t} B$ as a function of $t$

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




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



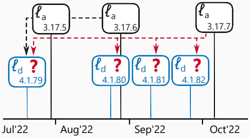




$t_A$ = 184 days
$t_B$ = 21 days

# Survival analysis on library update

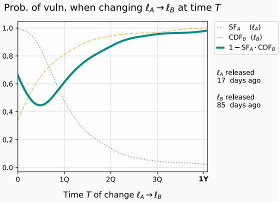
$A \xrightarrow{t} B$ means that we change from dependency $\ell_A$ to $\ell_B$ in $t$ time units counting from $t_0$ (“today”).

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

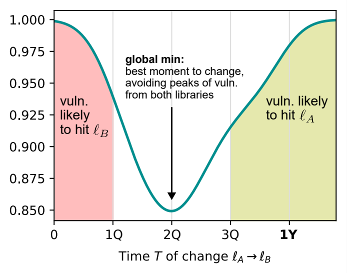


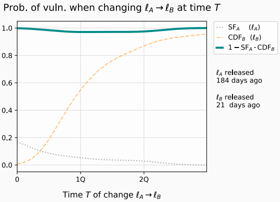

**Q:** $\mathsf{Pr}_{A,B}(t)$ = probability of vuln. of $A \xrightarrow{t} B$ as a function of $t$

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




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



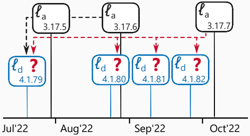




$t_A$ = 184 days
$t_B$ = 21 days

# Survival analysis on library update

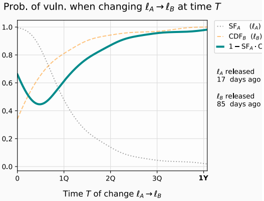
$A \xrightarrow{t} B$ means that we change from dependency $\ell_A$ to $\ell_B$ in $t$ time units counting from $t_0$ ("today").

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

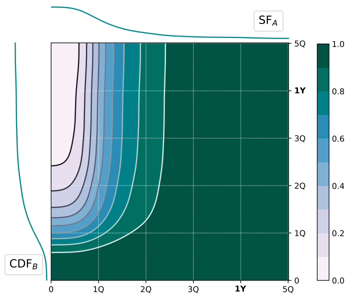


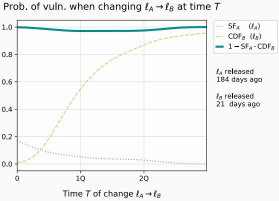

**Q:** $\mathsf{Pr}_{A,B}(t)$ = probability of vuln. of $A \xrightarrow{t} B$ as a function of $t$

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




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



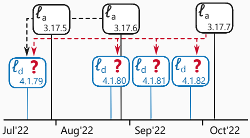




$t_A = 184$ days
$t_B = 21$ days

# Survival analysis on library update

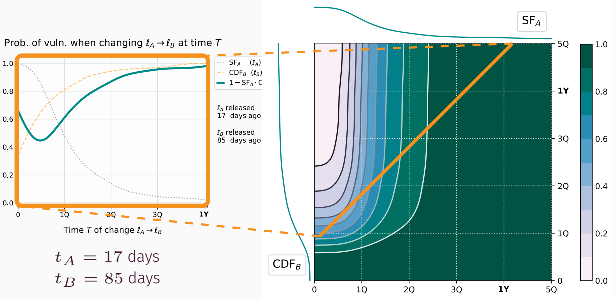
$A \xrightarrow{t} B$ means that we change from dependency $\ell_A$ to $\ell_B$ in $t$ time units counting from $t_0$ ("today").

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

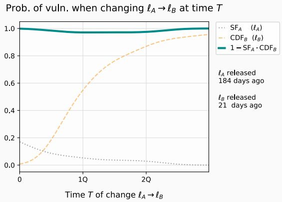



**Q:** $\mathsf{Pr}_{A,B}(t)$ = probability of vuln. of $A \xrightarrow{t} B$ as a function of $t$

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




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

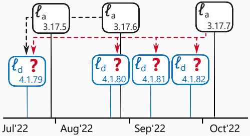



$t_A$ = 184 days
$t_B$ = 21 days

# Survival analysis on library update

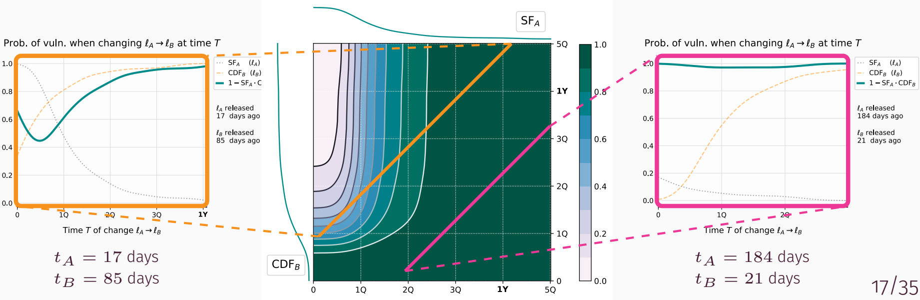
$A \xrightarrow{t} B$ means that we change from dependency $\ell_A$ to $\ell_B$ in $t$ time units counting from $t_0$ ("today").

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




**Q:** $\text{Pr}_{A,B}(t)$ = probability of vuln. of $A \xrightarrow{t} B$ 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 $\Delta t_x \doteq |t_x - t_0|$




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

$t_A$ = 184 days
$t_B$ = 21 days

# Vulnerabilities from any dependency

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

# Vulnerabilities from any dependency

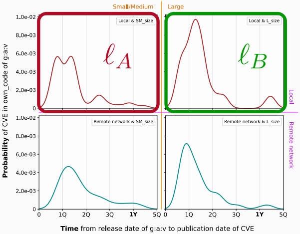
**Q:** $\mathsf{Pr}_{A,B}(t) =$ probability of vuln. in $\ell_A$ or $\ell_B$ before $t$

**A:** $\mathsf{Pr}_{A,B}(t) = \mathsf{Pr}(\min(\ell_A, \ell_B) \leqslant t) = 1 - (1 - \mathsf{Pr}_A(t))(1 - \mathsf{Pr}_B(t))$

# Vulnerabilities from any dependency

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

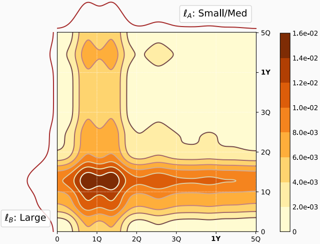
**A:** $\mathsf{Pr}_{A,B}(t) = \mathsf{Pr}(\min(\ell_A, \ell_B) \leqslant t) = 1 - (1 - \mathsf{Pr}_A(t))(1 - \mathsf{Pr}_B(t))$

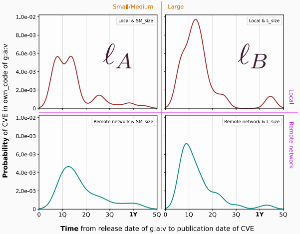


# Vulnerabilities from any dependency

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

**A:** $\text{Pr}_{A,B}(t) = \text{Pr}(\min(\ell_A, \ell_B) \leqslant t) = 1 - (1 - \text{Pr}_A(t))(1 - \text{Pr}_B(t))$

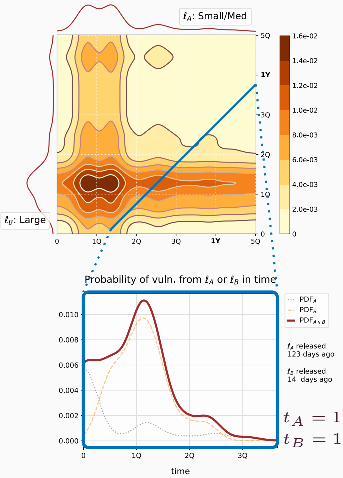


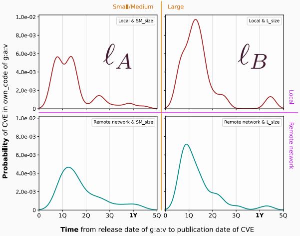
# Vulnerabilities from any dependency

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

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



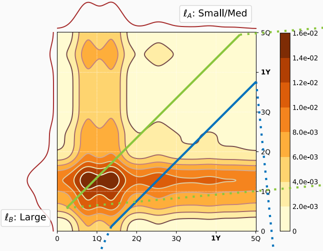

$t_A = 123$ days
$t_B = 14$ days

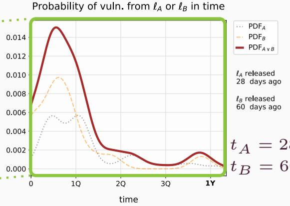


# Vulnerabilities from any dependency

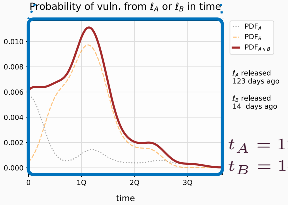
**Q:** $\Pr_{A,B}(t) =$ probability of vuln. in $\ell_A$ or $\ell_B$ before $t$

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

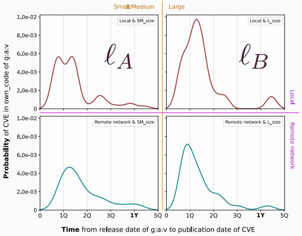



$t_A = 28$ days
$t_B = 60$ days



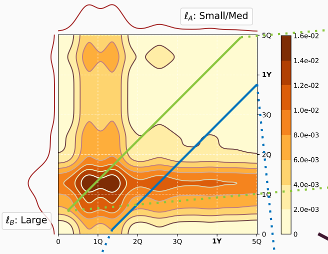

$t_A = 123$ days
$t_B = 14$ days

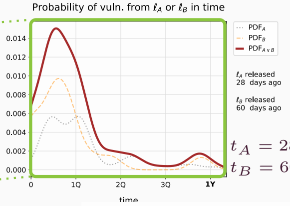


# Vulnerabilities from any dependency

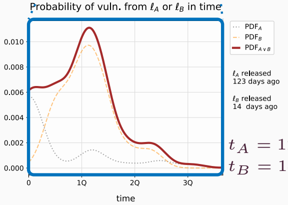
**Q:** $\Pr_{A,B}(t) =$ probability of vuln. in $\ell_A$ or $\ell_B$ before $t$

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

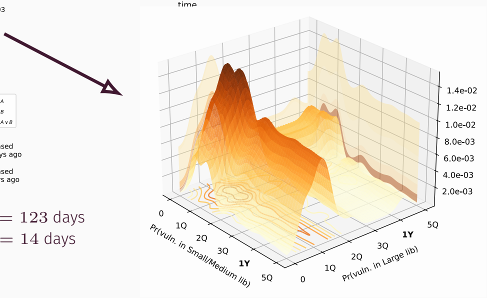



$t_A = 28$ days
$t_B = 60$ days



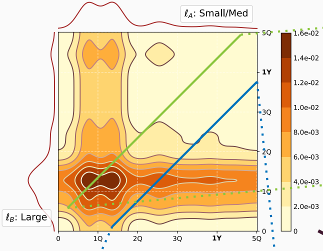

$t_A = 123$ days
$t_B = 14$ days

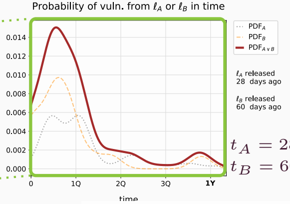


# Vulnerabilities from any dependency

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

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

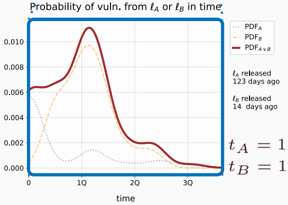


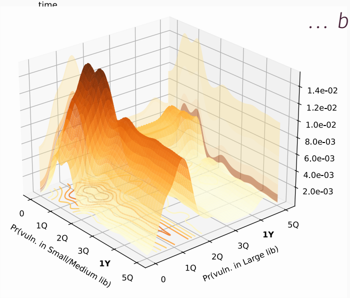

$t_A = 28$ days
$t_B = 60$ days

Nice for **2** dependencies!

*... btw I have 2000*



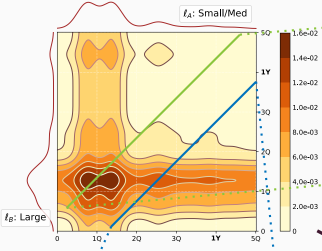

$t_A = 123$ days
$t_B = 14$ days

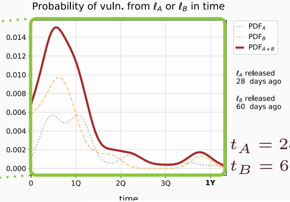


# Vulnerabilities from any dependency

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

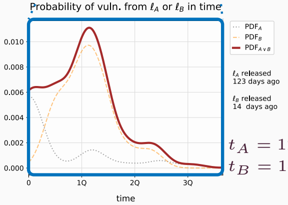
**A:** $\Pr_{A,B}(t) = \Pr(\min(\ell_A, \ell_B) \leqslant t) = 1 - (1 - \Pr_A(t))(1 - \Pr_B(t))$



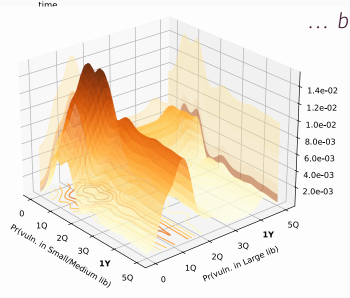




$t_A$ = 28 days
$t_B$ = 60 days

Nice for **2** dependencies!




$t_A$ = 123 days
$t_B$ = 14 days



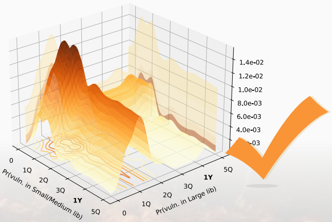

*… btw I have 2000*

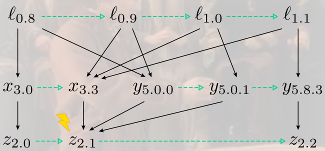
**TDTs!**

# Forecast model



CVE root-lib PDFs



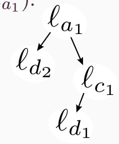




Time Dependency Trees

# Time Dependency Trees

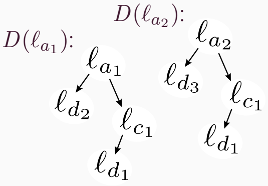
Dependency Trees in time

$D(\ell_{a_1})$:



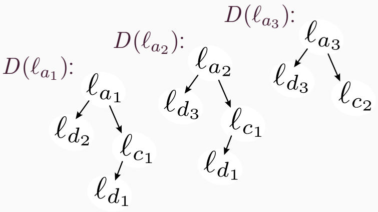
# Time Dependency Trees

Dependency Trees in time



# Time Dependency Trees

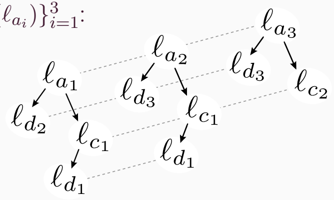
Dependency Trees in time



# Time Dependency Trees

Dependency Trees in time

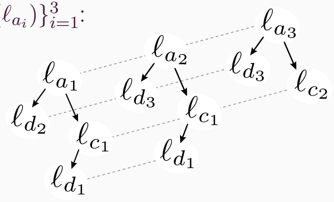
$\{D(\ell_{a_i})\}_{i=1}^{3}$:



# Time Dependency Trees

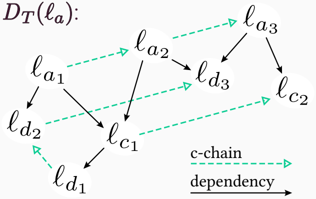
## Dependency Trees in time

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




## Time Dependency Tree

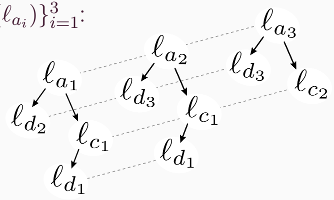
$D_T(\ell_a)$:



# Time Dependency Trees

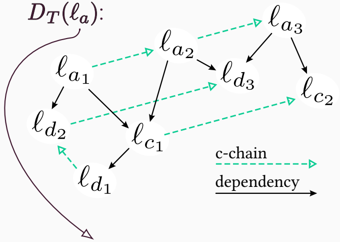
## Dependency Trees in time

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




## Time Dependency Tree

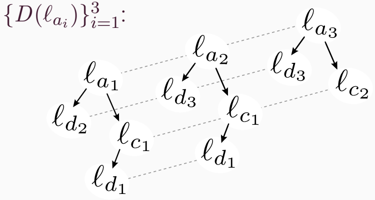
$D_T(\ell_a)$:



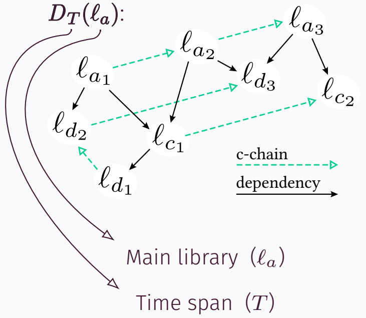

Main library ($\ell_a$)

# Time Dependency Trees

Dependency Trees in time



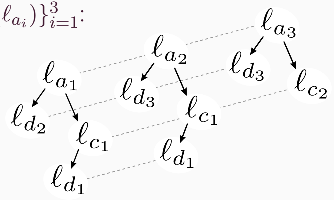

Time Dependency Tree



# Time Dependency Trees

Dependency Trees in time

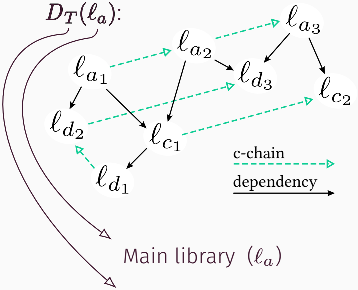
$\{D(\ell_{a_i})\}_{i=1}^{3}$:




$D_t(\ell_a) = D(\ell_{a_1})$
for any time point $t \in T$
after the release of $\ell_{a_1}$ and
before the release of $\ell_{a_2}$

Time Dependency Tree

$D_T(\ell_a)$:




Main library $(\ell_a)$

Time span $(T)$

# 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** $D_T(\ell)\big|_d$ yields *all instances* of dependency $d$ during time $T$

# Properties of TDT $D_T(\ell)$

## Theoretical

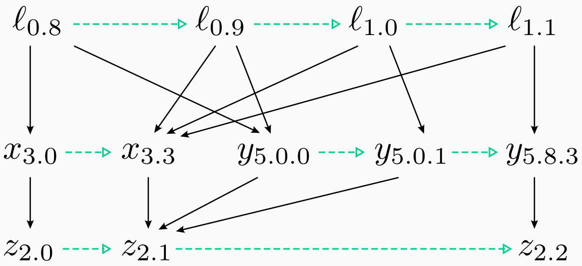
- Minimal graph representation (no lib-version repetition)
- Canonical for library $\ell$ and time span $T$
- Natural lifting of dependency trees to time

## Practical

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

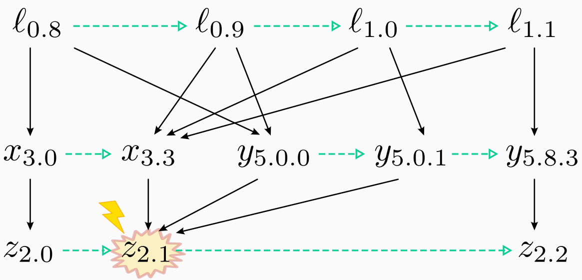
# SPoF in time and dependencies

My personal project uses $\ell_{1.0}$



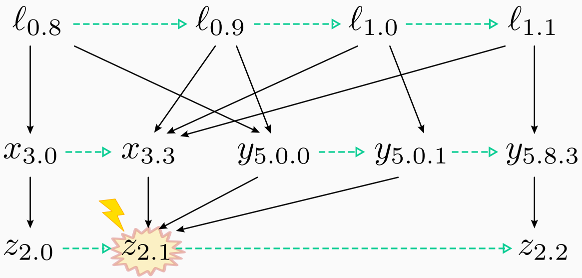
# SPoF in time and dependencies

My personal project uses $\ell_{1.0}$



# SPoF in time and dependencies

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** $D_T(\ell)\big|_d$ 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

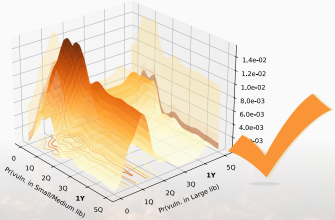
## Practical

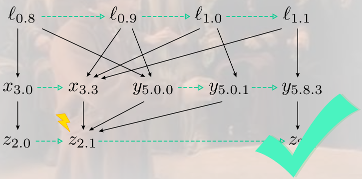
- **Time-indexing** $D_t(\ell)$ yields the dep. tree at time $t \in T$
- **Library-slicing** $D_T(\ell)\big|_d$ yields *all instances* of dependency $d$ during time $T$
- Reachability analysis can spot single-points-of-failure
- Can measure health/risk of development environment

# Forecast model



CVE root-lib PDFs




Time Dependency Trees

# 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

# Some things done

to be

# Some things ^to be^ done

- Other metrics to clusterise libraries for PDF-fitting

# Some things done

to be

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

# 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

# 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?

# References i


Henrique Alves, Baldoino Fonseca, and Nuno Antunes.
**Software metrics and security vulnerabilities: Dataset and exploratory study.**
In *EDCC*, pages 37–44. IEEE, 2016.

Junaid Akram and Ping Luo.
**SQVDT: A scalable quantitative vulnerability detection technique for source code security assessment.**
*Software: Practice and Experience*, 51(2):294–318, 2021.

Manar Alohaly and Hassan Takabi.
**When do changes induce software vulnerabilities?**
In *CIC*, pages 59–66. IEEE, 2017.

Amiangshu Bosu, Jeffrey C. Carver, Munawar Hafiz, Patrick Hilley, and Derek Janni.
**Identifying the characteristics of vulnerable code changes: An empirical study.**
In *FSE*, pages 257–268. ACM, 2014.

Zeki Bilgin, Mehmet Akif Ersoy, Elif Ustundag Soykan, Emrah Tomur, Pinar Çomak, and Leyli Karaçay.
**Vulnerability prediction from source code using machine learning.**
*IEEE Access*, 8:150672–150684, 2020.

Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray.
**Deep learning based vulnerability detection: Are we there yet.**
*IEEE Transactions on Software Engineering*, 48(9):3280–3296, 2021.

Istehad Chowdhury and Mohammad Zulkernine.
**Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities.**
*Journal of Systems Architecture*, 57(3):294–313, 2011.

Sundarakrishnan Ganesh, Tobias Ohlsson, and Francis Palma.
**Predicting security vulnerabilities using source code metrics.**
In *SweDS*, pages 1–7. IEEE, 2021.

Seulbae Kim, Seunghoon Woo, Heejo Lee, and Hakjoo Oh.
**VUDDY: A scalable approach for vulnerable code clone discovery.**
In *SP*, pages 595–614. IEEE, 2017.

David Last.
**Forecasting zero-day vulnerabilities.**
In *CISRC*, pages 1–4. ACM, 2016.

Chengwei Liu, Sen Chen, Lingling Fan, Bihuan Chen, Yang Liu, and Xin Peng.
**Demystifying the vulnerability propagation and its evolution via dependency trees in the NPM ecosystem.**
In *ICSE*, pages 672–684. ACM, 2022.

Hongzhe Li, Hyuckmin Kwon, Jonghoon Kwon, and Heejo Lee.
**A scalable approach for vulnerability discovery based on security patches.**
In *ATIS*, volume 490 of *CCIS*, pages 109–122. Springer, 2014.

Éireann Leverett, Matilda Rhode, and Adam Wedgbury.
**Vulnerability forecasting: Theory and practice.**
*Digital Threats*, 3(4):42:1–42:27, 2022.

Qiang Li, Jinke Song, Dawei Tan, Haining Wang, and Jiqiang Liu.
**PDGraph: A large-scale empirical study on project dependency of security vulnerabilities.**
In *DSN*, pages 161–173. IEEE, 2021.

Yi Li, Aashish Yadavally, Jiaxing Zhang, Shaohua Wang, and Tien N. Nguyen.
**Commit-level, neural vulnerability detection and assessment.**
In *FSE*, pages 1024–1036. ACM, 2023.

Andrew Meneely, Harshavardhan Srinivasan, Ayemi Musa, Alberto Rodríguez Tejeda, Matthew Mokary, and Brian Spates.
**When a patch goes bad: Exploring the properties of vulnerability-contributing commits.**
In *ESEM*, pages 65–74. IEEE, 2013.

Andrew Meneely and Laurie Williams.
**Strengthening the empirical analysis of the relationship between Linus' law and software security.**
In *ESEM*. ACM, 2010.

# References v


Ivan Pashchenko, Henrik Plate, Serena Elisa Ponta, Antonino Sabetta, and Fabio Massacci.
**Vuln4Real: A methodology for counting actually vulnerable dependencies.**
*IEEE Transactions on Software Engineering*, 48(5):1592–1609, 2022.

Gede Artha Azriadi Prana, Abhishek Sharma, Lwin Khin Shar, Darius Foo, Andrew E. Santosa, Asankhaya Sharma, and David Lo.
**Out of sight, out of mind? how vulnerable dependencies affect open-source projects.**
*Empirical Software Engineering*, 26(4), 2021.

Yaman Roumani, Joseph K. Nwankpa, and Yazan F. Roumani.
**Time series modeling of vulnerabilities.**
*Computers & Security*, 51:32–40, 2015.

Kazi Zakia Sultana, Vaibhav Anu, and Tai-Yin Chong.
**Using software metrics for predicting vulnerable classes and methods in Java projects: A machine learning approach.**
*Journal of Software: Evolution and Process*, 33(3), 2021.

Kazi Zakia Sultana, Ajay Deo, and Byron J. Williams.
**Correlation analysis among Java nano-patterns and software vulnerabilities.**
In *HASE*, pages 69–76. IEEE, 2017.

Nahid Shahmehri, Amel Mammar, Edgardo Montes de Oca, David Byers, Ana Cavalli, Shanai Ardi, and Willy Jimenez.
**An advanced approach for modeling and detecting software vulnerabilities.**
*Information and Software Technology*, 54(9):997–1013, 2012.

Yonghee Shin, Andrew Meneely, Laurie Williams, and Jason A. Osborne.
**Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities.**
*IEEE Transactions on Software Engineering*, 37(6):772–787, 2011.

Kazi Zakia Sultana and Byron J. Williams.
**Evaluating micro patterns and software metrics in vulnerability prediction.**
In *SoftwareMining*, pages 40–47. IEEE, 2017.

Huanting Wang, Zhanyong Tang, Shin Hwei Tan, Jie Wang, Yuzhe Liu, Hejun Fang, Chunwei Xia, and Zheng Wang.
**Combining structured static code information and dynamic symbolic traces for software vulnerability prediction.**
In *ICSE*, pages 169:1–169:13. ACM, 2024.

Emrah Yasasin, Julian Prester, Gerit Wagner, and Guido Schryen.
**Forecasting IT security vulnerabilities – an empirical analysis.**
*Computers & Security*, 88, 2020.

# Forecasting software vulnerabilities

## Probability Density Functions and Time Dependency Trees

C.E. Budde R. Paramitha F. Massacci

14th March 2024

ProSVED final event symposium