Turun yliopisto
University of Turku

Toward Validation of Textual Information Retrieval Techniques for Software Weaknesses

Jukka Ruohonen

Ville Leppänen

TIR 2018, Regensburg

Outline

1. Motivation
2. Materials
3. Methods
4. Results
5. Conclusion

Motivation

- Software **vulnerabilities** are concrete software bugs with security implications (usually cataloged with CVEs)
- Software **weaknesses** are abstractions for the underlying "root causes" behind vulnerabilities (usually cataloged with CWEs)
- **Challenges** for archiving of vulnerabilities and weaknesses
  - Proliferation of databases & reporting infrastructures, etc.
- Many vulnerability databases **do not catalog weaknesses**
  - Manual work required, complexity of the CWE standard, etc.
- Thus, **automation** is desirable for both research and practice

Materials (1/5)

- Three data sources:

1. NIST
   Information Technology Laboratory
   NATIONAL VULNERABILITY DATABASE
   NVD
2. CWE Common Weakness Enumeration
   A Community-Developed List of Software Weakness Types
3. snyk Test Features Vulnerability DB Blog Partners Pricing Docs About
   Vulnerability DB
   Detailed information and remediation guidance for known vulnerabilities.

Materials (2/5)

- The **goal** is to map vulnerability entries in the Snyk database into weaknesses in the CWE database, using a subset of weaknesses that have vulnerability entries in NVD
  - One-to-one mappings between Snyk and the CWE database
  - NVD has only an indirect role to map CVEs to CWEs
  - *Maven*, *pip*, *npm*, and *RubyGems* are included from Snyk
- Mappings can be "direct" (via CWE) or "indirect" (via CVE)
- In addition to the primary "first-order" textual data in the Snyk database, the content behind the **online references** provided are used as additional "second-order" textual data

Materials (3/5)

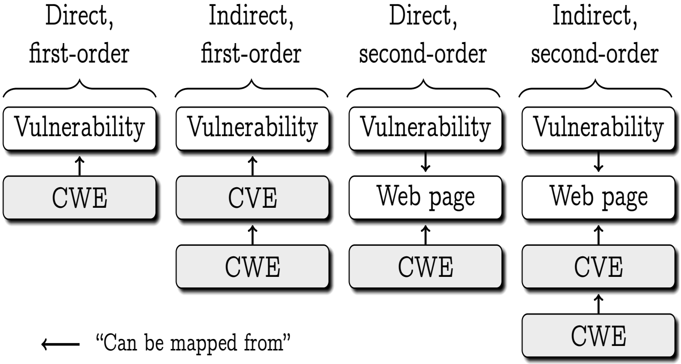

Figure: An Example of Four Abstract Relations for Software Weaknesses

Materials (4/5)

```
## Overview
[`pymongo`](https://pypi.python.org/pypi/pymongo) is a Python driver for
MongoDB.

`bson/_cbsonmodule.c` in the mongo-python-driver (aka.  pymongo) before
2.5.2, as used in MongoDB, allows context-dependent attackers to cause a
denial of service (NULL pointer dereference and crash) via vectors related
to decoding of an "invalid DBRef."

## References
- [NVD](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-2132)
- [Github Commit](https://github.com/mongodb/mongo-python-driver/commit/
a060c15ef87e0f0e72974c7c0e57fe811bbd06a2)
```

Materials (5/5)

[...] The software performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer. Certain languages allow direct addressing of memory locations and do not automatically ensure that these locations are valid for the memory buffer that is being referenced. This can cause read or write operations to be performed on memory locations that may be associated with other variables, data structures, or internal program data. As a result, an attacker may be able to execute arbitrary code, alter the intended control flow, read sensitive information, or cause the system to crash.

The generic term memory corruption is often used to describe the consequences of writing to memory outside the bounds of a buffer, when the root cause is something other than a sequential copies of excessive data from a fixed starting location (i.e., classic buffer overflows or CWE-120). This may include issues such as incorrect pointer arithmetic, accessing invalid pointers due to incomplete initialization or memory release, etc. [...]

Methods (1/2)

- The information retrieval techniques used are compared against commonly used **regular expression searches**
  - Estimation is carried out with a subset within which each entry matched the regular expression searches
  - $n_1 = 82$ weaknesses and $n_2 = 585$ vulnerabilities
- **Precision** $= \frac{(\#\ \text{same CWE})}{(\#\ \text{same CWE}) + (\#\ \text{different CWE})}$
  - Do not connote with "true positives" and "false positives"

Methods (2/2)

- A fairly typical **pre-processing** routine (i.e., lower-casing, tokenization, trimming, stemming, etc.) is used
- Analysis carried out with **unigrams**, **bigrams**, and **trigrams**
- Five different **weights** are used: (1) term (i.e., *n*-gram) frequency (TF), (2) TF-LOG, (3) TF-BOOLEAN, (4) TF-IDF, and (4) DLM-IDF (document length normalization with IDF)
- **Cosine similarity** used as the similarity metric
  - Maximum values are used to pick CWEs for vulnerabilities
- In addition, the so-called **latent semantic analysis** (LSA) was briefly examined as an additional validation check

Results (1/4)

Table: Descriptive Statistics

	Unigrams	Bigrams	Trigrams
Unique *n*-grams	8435	32166	31745
Average document length	1095	935	839
• Average CWE length	424	357	252
• Average vulnerability length	1175	1016	921

Results (2/4)

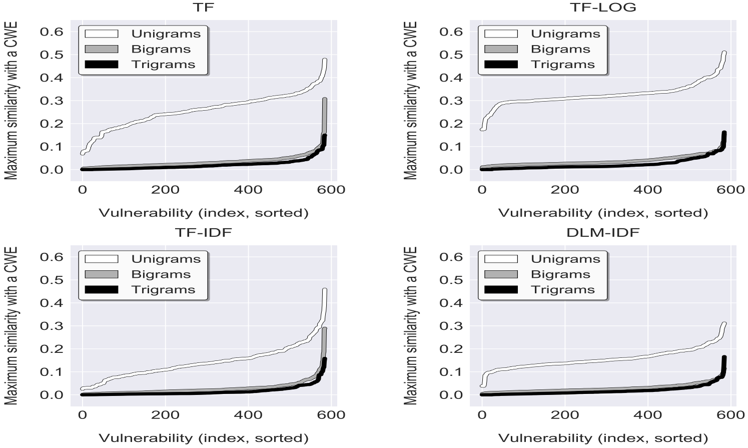

Figure: Maximum Cosine Similarities According to Four Weights

Results (3/4)

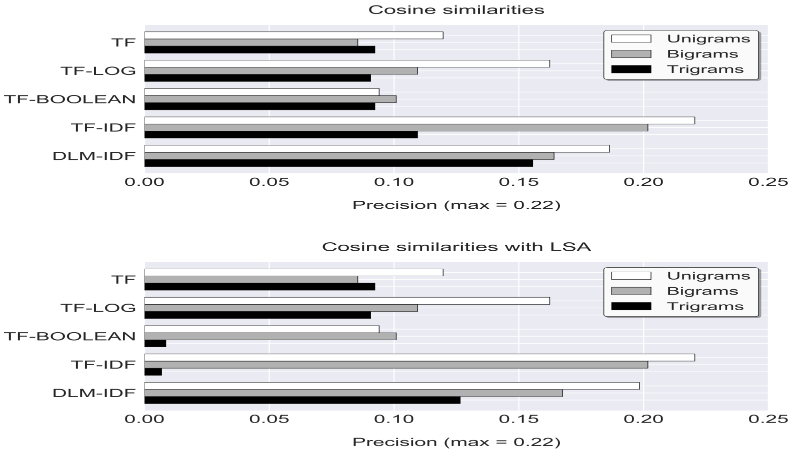

Figure: Precision with Five Weights

Results (4/4)

Table: Average per-Repository Precision (TF-IDF)

	Maven	*pip*	*npm*	*RubyGems*
Unigrams	0.17	0.34	0.55	0.25
Bigrams	0.16	0.31	0.09	0.62
Trigrams	0.10	0.12	< 0.01	0.50

Conclusion

- Common information retrieval techniques **perform poorly**

  $\implies$ Recommendation for practitioners: whenever possible, explicitly reference database entries with CVEs/CWEs

  $\implies$ Further validation work is required, however

- Two main possibilities for moving forward:
  1. Data **enrichment** $\implies$ reference corpora for security
  2. Data **enlargement** $\implies$ toward big data with web crawling

Thank you

Questions?