A Clairvoyant Approach to Evaluating Software (In)security

Bhushan Jain, Chia-Che Tsai*, Don Porter
Which is More Secure?

Can we evaluate security empirically?
Most just use LoC and #CVE reports as prediction of security.
Is it a Good Idea to Use Lines of Code?

- Conventional wisdom:
  - # of LoC $\Rightarrow$ # of bugs
  - Easy to formally verify or code review small LoC

- “There are, on the average, about 21 bugs per KLoC discoverable” [Gaffney, TOSE '84]

- “Commercial software typically has 20 to 30 bugs for every 1,000 lines of code”
  
  —CMU’s CyLab quoted by WIRED magazine in 2004

LoC seems logical way to predict security problems
Is LoC Correlated to #(Vulnerabilities)?

- 164 applications
  - Open-source
  - 5-20 years in CVE DB
  - Primary language:
    - C: 126
    - C++: 20
    - Python: 6
    - Java: 12

Log10

Low LoC = low #CVE
High LoC = High #CVE

LoC not significant within orders of magnitude
LoC not a reliable predictor of vulnerabilities
May be we can try program complexity?

- Conventional wisdom:
  - Complex program => high probability of vulnerabilities

- Cyclomatic Complexity
  [McCabe, TOSE '76]:
  # of linearly independent paths

 Complexity too is noisy within orders of magnitude

Complexity not necessarily correlated to the #CVE reports
Other Conventional Wisdom

- Large attack surface ⇒ more opportunities for attacker
  - Relative Attack Surface Quotient (RASQ) [Howard et al., 2005]
  - Resources, communication channels, access rights for attackers
  - Specific to configuration

- Secure design guidelines ⇒ less # of vulnerabilities
  - Design Security Standards
    - NIST 800-55, Common Criteria, ISO/IEC 27004
  - Qualitative, subjective, no precise evaluation model

These wisdom are mostly qualitative
## Code Properties Reveal Security Aspects

<table>
<thead>
<tr>
<th>Code Properties</th>
<th>Security Aspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choice of language</td>
<td>Safety of languages &amp; runtimes</td>
</tr>
<tr>
<td>Lines of code</td>
<td>Difficulty of code-checking/verification</td>
</tr>
<tr>
<td>Cyclomatic complexity</td>
<td>Variant of execution paths</td>
</tr>
<tr>
<td>Attack surface</td>
<td>Number of paths to attack</td>
</tr>
</tbody>
</table>

Weighted aggregation covers more security aspects
Code properties in isolation doesn’t evaluate security. Aggregation may help.
Ideal Security Evaluation

- Predict risk of compromise
  - Attacker effort (qualitative)
  - Vulnerabilities (quantitative)
- Help improve code over versions
- Improved code = Improved metric score
- Compare similar software

Predict # and severity of *all* vulnerabilities
Can We Just Predict Bugs Instead?

➢ “Many security holes in software are the result of software bugs...”

— Seth Hallem, CEO of Coverity, 2004

➢ Vast research predict bugs based on code properties
  
  – A weighted correlation of code properties and bugs
  
  – Too many false positives
  
  – Need human intervention

Maybe #bugs is a good way of predicting vulnerabilities
Bugs and Vulnerabilities: It’s Complicated!

- Bugs don’t foreshadow vulnerabilities
  - Study [Camilo et al., MSR '15]: # of bugs ↢ # of vulnerabilities
  - Buggiest files ≠ files with many vulnerabilities

- Code properties may have different relation to vulnerabilities
  - Study [Shin et al., TOSE '11]: some code properties are indicative
    - #functions, #declarations, #preprocessing lines, #branches, #input and output arguments to a function

Vulnerabilities may correlate with code properties differently
Let’s Learn the Correlation

➢ Hypothesis:
  – Machine learnable correlation between code properties & vulnerabilities

https://imgs.xkcd.com/comics/correlation.png

May not be perfect, but we have to do SOMETHING
What Do We Need?

- Large ground truth data
  - More than 80,000 vulnerabilities in 400 applications and systems
- Representative data
  - #CVE Reports vary based on maturity and attention received
- May be missing security-indicative code properties
  - Any suggestions are most welcome!
Calculating Other Code Properties

- Data flow analysis
  - # of expressions, functions, data structures

- Control flow analysis
  - # of calling and return targets

- Abstract interpretation
  - # of paths triggered by specific range of inputs

Static analysis can help collect more properties
Vulnerability Information

CVE-2016-8740 Detail

Impact

- CVSS v3 Base Score: 7.5
- Impact Score: 3.6
- Exploitability Score: 3.9
- Attack Vector (AV): Network
- Attack Complexity (AC): Low
- Privileges Required (PR): None
- User Interaction (UI): None
- Scope (S): Unchanged
- Confidentiality (C): None
- Integrity (I): None
- Availability (A): High

Vulnerability Type

- Input Validation (CWE-20)
- Resource Management Errors (CWE-399)

Root causes

Configuration

- cpe:2.3:a:apache:http_server:2.4.17:*
- cpe:2.3:a:apache:http_server:2.4.18:*
- cpe:2.3:a:apache:http_server:2.4.19:*
- cpe:2.3:a:apache:http_server:2.4.20:*
- cpe:2.3:a:apache:http_server:2.4.21:*

Affected versions

A vector of information available from CVE reports
System Proposal

 CVE database

 App 1
 App 2
 App N

 Selecting ground truth: Applications with converging CVE history
System Proposal

CVE database

App 1

App 2

App N

Static Analysis Tools

(Lang, LoC, Cyclo, RASQ, ...)

(Lang, LoC, Cyclo, RASQ, ...)

(Lang, LoC, Cyclo, RASQ, ...)

Automated testbed to collect a vector of code properties
System Proposal

Static Analysis Tools

CVE database

App 1

App 2

App N

ML (w/ cross validation)

(Lang,LoC,Cyclo,RASQ,...)

AV=(N,A,L,P),AC=...

AV=(N,A,L,P),AC=...

AV=(N,A,L,P),AC=...

Classifiers

Mining correlation between code properties & vulnerability classes
System Proposal

CVE database

App 1
App 2
App N

Static Analysis Tools

ML (w/ cross validation)
(Lang, LoC, Cyclo, RASQ, ...)
AV=(N, A, L, P), AC= ... AV=(N, A, L, P), AC= ... AV=(N, A, L, P), AC= ... AV=(N, A, L, P), AC= ... AV=(N, A, L, P), AC= ...

The eval result is prediction of vulnerability severity & classes

Classifiers predict #, severity, and classes of vulnerabilities
Oh No! Not Another Security Metric!

- Easily extendable
- Can only improve with time (more CVE data)
- Doesn’t rely on only one code property
- Gives useful feedback to developers

Supposed to be the one metric to rule them all!

https://imgs.xkcd.com/comics/standards.png
Using the Metric

we propose to build a series of classifiers for SW vulnerability:

$$
E \left[ AV \ (\text{Attack Vector}) = N \ (\text{Network}) \right] = 
\text{Lang} \times W_0 + \log_{10}(\text{LoC}) \times W_1 + \text{Cyclo} \times W_2 + \text{RASQ} \times W_3 + \ldots
$$

- Confirm or update conventional wisdom
- Balance multiple properties
- Hint possible security enhancement:
  - Defenses against potential attacks
  - Improve code property

More than just another “security score”!

Metric can be integrated with regression testing
Conclusion

- LoC, complexity, other metrics are noisy
- We propose to approximate risk of having a vulnerability
- Learn weighted relation of code properties to vulnerabilities
- Challenge:
  - Extract meaning from incomplete ground truth

Bhushan Jain
bhushan@cs.unc.edu