Can Knowledge of Technical Debt Help Identify Software Vulnerabilities?

Robert L. Nord, Ipek Ozkaya, Edward J. Schwartz, Forrest Shull, Rick Kazman
Copyright 2016 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM-0003863
Security Issues

10977: Crash due to large negative number.

"We could just fend off negative numbers near the crash site or we can dig deeper and find out how this -10000 is happening."

"Time permitting, I'm inclined to want to know the root cause. My sense is that if we patch it here, it will pop-up somewhere else later."

“There have been 28 reports from 7 clients... 18 reports from 6 clients”

“hmm ... reopening. the test case crashes a debug build, but not the production build. I have confirmed that the original source code does crash the production build, so there must be multiple things going on here.”
What are we measuring?

**defect** – error in coding or logic that causes a program to malfunction or to produce incorrect/unexpected results

**technical debt** – design or implementation construct traced to several locations in the system, that make future changes more costly

**technical debt** – design or implementation construct traced to several locations in the system, that make future changes more costly

**vulnerability** – system weakness in the intersection of three elements:
- system flaw,
- attacker access to the flaw,
- attacker capability to exploit the flaw

Can Knowledge of Technical Debt Help Identify Software Vulnerabilities?

August 8, 2016

© 2016 Carnegie Mellon University

[DISTRIBUTION STATEMENT A]
This material has been approved for public release and unlimited distribution.

Defects are not technical debt, but technical debt as it lingers in the system increases defect proneness.

Similarly, technical debt increases vulnerability risks.

Not all defects are vulnerabilities, but defect proneness does imply increased vulnerability risks.

Some issues just overlap, making it hard to tease apart!
Research question

Are software components with accrued technical debt more likely to be vulnerability-prone?
Data Set

Chromium version: 17.0.963.46

Released: February 8, 2012

Files: 18,730;
11k files with bugs
289 files with vulnerabilities

Issue range: Feb 1, 2010 – Feb 8, 2012

Issues: #bug: 14k;
#security: 79

Chromium project

• Began in 2008
• Complex web-based application that operates on sensitive information and allows untrusted input from both web clients and servers.
**Approach**

**Identify software vulnerabilities**
- security label
- identify indicator from issue: CWE
- trace to file

**Identify technical debt**
- apply classification rules to issues
- extract design problem and rework from issues
- trace to file
- indicator from file: bugs and churn
- indicator from file: design flaws

**Model relationships**
- design concepts
- technical debt indicators

Test for correlations between technical debt prone files and files with known vulnerabilities.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td></td>
</tr>
<tr>
<td>Priority</td>
<td></td>
</tr>
<tr>
<td>Label: Security, Impact, Severity</td>
<td></td>
</tr>
<tr>
<td>Type: Bug, Bug-Security</td>
<td></td>
</tr>
<tr>
<td>CVE</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Commit History</th>
<th>Code File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue</td>
<td>Name</td>
</tr>
<tr>
<td>Code</td>
<td>LOC</td>
</tr>
<tr>
<td>Version history</td>
<td>Age</td>
</tr>
</tbody>
</table>
Technical debt

In software-intensive systems, technical debt is a software design issue that:

- Exists in an **executable system artifact**, such as code, build scripts, data model, automated test suites;
- Is traced to **several locations** in the system, implying issues are not isolated but propagate throughout the system artifacts.
- Has a **quantifiable** effect on system attributes of interest to developers (e.g., increasing defects, negative change in maintainability and code quality indicators).
Can Knowledge of Technical Debt Help Identify Software Vulnerabilities?

August 8, 2016

© 2016 Carnegie Mellon University

DISTRIBUTION STATEMENT A
This material has been approved for public release and unlimited distribution.


---

**Crash due to large negative number.**

We could just fend off negative numbers near the crash site or we can dig deeper.

Time permitting, I'm inclined to want to know the root cause.

I have confirmed that the original source code does crash the production build, so there must be multiple things going on here.

---

There have been 28 reports from 7 clients... 18 reports from 6 clients.

My sense is that if we patch it here, it will pop-up somewhere else later.

hmm ... reopening. The test case crashes a debug build, but not the production build.

---

21 of 79 issues labeled security are classified as technical debt.
Finding: Developers Use Technical Debt Concepts

Developers addressing security issues are using technical debt related concepts (italicized):

- getting to the root cause
- understanding the underlying design issues
- recording symptoms where changes are taking longer than usual or problems are reoccurring
- predicting consequences in the longer term
- building evidence for a more substantial fix
Indicator: Design flaws

Unstable Interface
Modularity Violation
Improper Inheritance
Cycle

## Unstable Interface

<table>
<thead>
<tr>
<th>Function</th>
<th>Dependencies</th>
<th>Commentary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ui.gfx.size.cc</td>
<td>Use,3,2,3,3,1,1,2</td>
<td>(1)</td>
</tr>
<tr>
<td>2 ui.gfx.size.h</td>
<td>Call,3,5,4,2,1,2,1,1</td>
<td>(2)</td>
</tr>
<tr>
<td>3 ui.gfx.point.h</td>
<td>Call,3,5,3,1,1,2,1,1</td>
<td>(3)</td>
</tr>
<tr>
<td>4 ui.gfx.rect.h</td>
<td>Call,3,Call,4,Call,5,Call,6,Call,6</td>
<td>(4)</td>
</tr>
<tr>
<td>5 ui.gfx.rect.cc</td>
<td>Call,3,Call,2,Call,3,Call,6,Call,6,Call,6,Call,6</td>
<td>(5)</td>
</tr>
<tr>
<td>6 webkit.plugins.ppapi.ppapi_plugin_instance.cc</td>
<td>Call,1,Call,Call,Call,2,Call,1,Call,1,Call,1</td>
<td>(6)</td>
</tr>
<tr>
<td>7 content.renderer.paint_aggregator.cc</td>
<td>Call,1,Call,1,Call,2,Call,1,Call,1,Call,5,Call,2,Call,2</td>
<td>(7)</td>
</tr>
<tr>
<td>8 content.renderer.render_widget.cc</td>
<td>Call,1,Call,2,Call,1,Call,2,Call,1,Call,1,Call,5,Call,2,Call,2</td>
<td>(8)</td>
</tr>
<tr>
<td>9 ui.gfx.rect_unittest.cc</td>
<td>2,Call,1,Call,5,Call,3,2,2,3,Call,3,2,2,3,Call,3,2,2,3</td>
<td>(9)</td>
</tr>
<tr>
<td>10 webkit.plugins.webview_plugin.cc</td>
<td>1,1,Call,2,1,2,2,1,2,2,1,2,2,1,2,2,1</td>
<td>(10)</td>
</tr>
<tr>
<td>11 ui.gfx.blit.cc</td>
<td>Call,Call,1,Call,2,Call,2,Call,2,Call,2,Call,2,Call,2</td>
<td>(11)</td>
</tr>
</tbody>
</table>
Modularity Violation

Shared secret between files
Should be extracted as design rules

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ContextConfig.java</td>
<td>1,31</td>
</tr>
<tr>
<td>2</td>
<td>TldConfig.java</td>
<td>31</td>
</tr>
</tbody>
</table>
Analysis: Design Flaws - 1

Increased rates of design flaws are strongly correlated with increased rates of security bugs.

<table>
<thead>
<tr>
<th>Project</th>
<th>Bug/Design Flaw Correlation</th>
<th>Change/Design Flaw Correlation</th>
<th>Sec Bug/Design Flaw Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chrome</td>
<td>0.987</td>
<td>0.988</td>
<td>0.979</td>
</tr>
</tbody>
</table>

Design flaws extracted using dependency analysis at the class level within files: unstable interface, modularity violation, improper inheritance, cycles.

Analysis: Design Flaws - 2

Moreover, being involved in more types of design flaws correlates with the presence of vulnerabilities.

<table>
<thead>
<tr>
<th># Types of Design Flaws</th>
<th>Non-vuln files</th>
<th>Vuln files</th>
<th>% have vulns.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8544</td>
<td>47</td>
<td>0.5%</td>
</tr>
<tr>
<td>1</td>
<td>7357</td>
<td>141</td>
<td>2%</td>
</tr>
<tr>
<td>2</td>
<td>2345</td>
<td>91</td>
<td>4%</td>
</tr>
<tr>
<td>3</td>
<td>194</td>
<td>10</td>
<td>5%</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>
Finding: Evidence of Correlation

Finding: We see evidence of correlations between vulnerabilities and technical debt indicators such as design flaws and code churn: the more types of design flaws a file is involved in, the higher the likelihood of it also having vulnerabilities; files with vulnerabilities tend to have more code churn.
Qualitative and Quantitative Analysis

Classifying TD from Issues labeled Security

<table>
<thead>
<tr>
<th></th>
<th>Not TD</th>
<th>TD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Flaws</td>
<td>50</td>
<td>15</td>
</tr>
<tr>
<td>No Design Flaws</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

79 issues are labeled security
- 21 are classified as technical debt
- 65 trace to files containing design flaws
## Design Flaws and Future Consequences

### Classifying TD from Issues labeled Security

<table>
<thead>
<tr>
<th>Not TD</th>
<th>TD</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

#### Design Flaws

- Detecting Design Flaws in Code
- No Design Flaws

#### Flaw Classification

- **Flaw:** Modularity violation
- **66931:** "Is it a workaround ... the root bug ... long term fix“
- **68766:** I've got my bandaid fix all reviewed and ready to check in once the tree reopens. But this problem sounds nasty enough that we definitely need a real fix.”

Flaws: modularity violation, cycle, improper inheritance.
### Partial Evidence

#### Classifying TD from Issues labeled Security

<table>
<thead>
<tr>
<th></th>
<th>Not TD</th>
<th>TD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defect</td>
<td>50</td>
<td>15</td>
</tr>
<tr>
<td>Feature</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Design Problem</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

#### Design Flaws

- **Detecting Design Flaws in Code**
  - No Design Flaws

### Flaw Examples

- **67577**: "This is a 2-liner. I'll take it, if only to get our rampant security bug list down by one.”
  - Flaw: modularity violation

- **64108**: “feature was never fully implemented, we may not have put in proper checks to prevent this.”
  - Flaws: modularity violation, cycle
### Supplement Static Analysis with Developer Knowledge

#### Classifying TD from Issues labeled Security

<table>
<thead>
<tr>
<th></th>
<th>Not TD</th>
<th>TD</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Design Flaws</td>
<td>50</td>
<td>15</td>
</tr>
<tr>
<td>Design Flaws</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

10977: “we could just fend off ... or we can dig deeper”  
“if we patch it here, it will pop-up somewhere else later”

70589: “My plan is to back out the brokenness, and fix it properly later”
Multiple Sources of Information

Our preliminary findings demonstrate that analysis necessitates using code, issue trackers, and commit history in concert.
Threats to Validity

Data quality and size
Manual inspection
Identification of technical debt and vulnerabilities
Summary / Future Work

Technical debt matters.

• **Finding 1:** When they address security issues, software developers use technical debt concepts to discuss design limitations and their consequences on future work.

• **Finding 2:** Correlations between vulnerabilities and technical debt indicators warrant further research.

Future work

• Experiment with modifiability and security taxonomies
• Apply to additional data sets
• Investigate causal links
• Codify as design flaws that tools can analyze