Secure Open Source Collaboration:  
An Empirical Study of Linus’ Law

“Many eyes make all bugs shallow”  
- Eric Raymond (Linus’ Law)

Andy Meneely

CSC 591-003

October 26, 2009
Motivation

- Software has vulnerabilities
  - Need to find them and fix them
  - Need to prevent them.

- People develop software
  - Behind every software project is a group of people, usually collaborating in teams.

- People work in large groups
  - All about communication & coordination
  - E.g. open source software
Linus’ Law

“Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone”

“Many eyes make all bugs shallow”
- Eric Raymond

Is this really true?
(Do the numbers match up?)
- More people → Too many cooks in the kitchen?
- Large project → Parts don’t get the focus they need?
Objectives

• **Goal:** to reduce security vulnerabilities
  – by providing *actionable insight*
  – into the *structural nature* of developer collaboration in open source software

**Specifically:**

- Many developers $\rightarrow$ Diverse developer perspectives $\rightarrow$ More secure
- Many developers $\rightarrow$ Lack of focus on certain components $\rightarrow$ Less secure
Case Study: Linux Kernel

• Red Hat Enterprise Linux 4 Kernel
  – Over 14,000 source code files (*.c, *.h, *.s)
  – Over 10,000 files changed in the 16 months prior to release
  – ~200 files found to have a post-release vulnerability
  – ~550 developers
## Vulnerability Data Collection

### Vulnerable Behavior

- **File**: `/fs/exec.c`
- **Metric**: TBD
- **# Vulns**: 1
- **Vulnerability**: Vulnerable

### Bugzilla

- **Can we reproduce?**
- **How should we fix?**
- **What does it affect?**

### Backport Patch

- **Upstream Commit**
- **Fixed in**

---

<table>
<thead>
<tr>
<th>File</th>
<th>Metric</th>
<th># Vulns</th>
<th>Vulnerability</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/fs/exec.c</code></td>
<td>TBD</td>
<td>1</td>
<td>Vulnerable</td>
</tr>
<tr>
<td><code>/fs/inode.c</code></td>
<td>TBD</td>
<td>0</td>
<td>Neutral</td>
</tr>
</tbody>
</table>
Observing People

- Focus: observe people *in groups*
  - Record of who worked on what: *version control change logs*
  - Study the *structure* of code change in terms of *people*

Laurie

changed 8/20, 10pm

/fs/exec.c

changed 8/21, 9am

Lucas
Observations to Metrics: NumDevs

• We examined four sub-hypotheses of Linus’ Law (*we’ll be presenting two today*)

\( H_{\text{NumDevs}} \): Files changed by many developers were less likely to have a vulnerability

\textit{NumDevs}: the number of distinct developers who made a commit to a given file
Observations to Metrics: Unfocused

\( H_{\text{Unfocused}} \): Files changed by developers who were working on many other files at the time were more likely to have a vulnerability (an “unfocused contribution”)

Take into account the other files that the contributing developers were working on
Linux Kernel Contribution Network

- **Blue** nodes are developers
- **Red** nodes are files
- Connections
  - Only if a developer changed a file
  - Thicker = many changes

- **Network analysis centrality**
  - Central to the network → unfocused contribution
Results

• $H_{\text{NumDevs}}$: Files changed by many developers were less likely to have a vulnerability
  
  A file was 16 times more likely to have a vulnerability if it had 9 or more developers who changed it (33% vs. 2%)

• $H_{\text{Unfocused}}$: Files with unfocused contributions are more likely to have security vulnerabilities

  Contribution Network centrality was higher ($p<0.01$) for files with vulnerabilities
Limitations

• Single case study, single release

• Metrics can only measure where we have developer activity
  – In the RHEL4 kernel, all vulns found were in files that were changed within 16 months of release

• Study focused on known vulnerabilities, not latent ones
Discussion

• Did we prove or disprove Linus’ Law? No, but…
  – Sometimes too many people is problematic
  – Committing to the community and focusing on a smaller group of files is better

• Questions for the future:
  – How many developers is ideal? (Does that generalize?)
  – Who (in terms of group structure) is ideal for working on code?
  – Can we improve task assignment with these metrics?
  – Can we measure many eyes, instead of many hands?
Thank you!

• Questions?