Where Should I Look? Using Metrics to Prioritize Vulnerability Removal Efforts

Laurie Williams

Department of Computer Science

North Carolina State University

Recognition

- Distinguished (Graduated) PhD Students
  - Andy Meneely (RIT)
  - Michael Gegick (US DoD)
  - Yonghee Shin (George Mason)
  - Nachi Nagappan (Microsoft Research)
- In Process PhD Student
  - Patrick Morrison
- Colleagues
  - Kim Herzig (Microsoft Research)
  - Brenden Murphy (Microsoft Research)
  - Tom Zimmerman (Microsoft Research)

Agenda

- Using metrics to predict the presence of security vulnerabilities in code
  - Static analysis alerts
  - Developer metrics
  - Complexity
  - Traditional code metrics (fault prediction)
- Misc observations

Vulnerability- and Attack-prone Components

Reliability context → Security context

Fault-prone component
Likely to contain faults
→
Vulnerability-prone component
Likely to contain vulnerabilities

Failure-prone component
Likely to cause failures
→
Attack-prone component
Likely to be exploited

Metrics – What are they good for?

- Prediction: We can use them to predict where vulnerabilities are and then prioritize our validation and verification efforts to those areas
- Change action: We can use them to change our behavior and our practices: "actionability"

General procedure

- Gather “internal” metrics about a product
- Gather discovered vulnerability data about a product
- Put the metrics into a statistical model: to look for correlations, predictions
- Validate model using a cross validation technique or with next release

- Vulnerability-prone component/file are those that have at least one vulnerability identified during testing or reported by customers or third-party researchers.

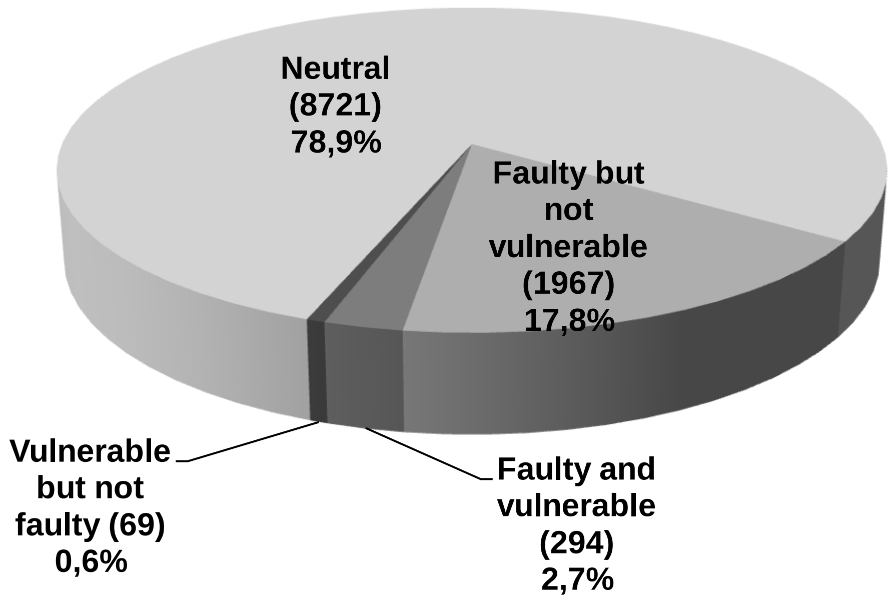
Threats to Validity / Challenges

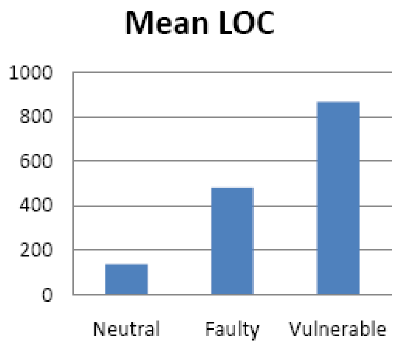
- Residual/latent vulnerabilities in software are possible.
- Vulnerability count is a function of security testing effort, customer usage, ease of attack, the attractiveness of the target, and <u>malicious intent</u>.
- Identified vulnerabilities are scarce.

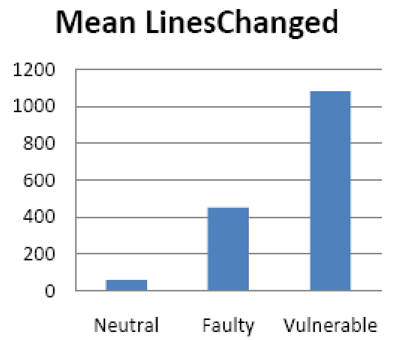
Software Engineering @ NCSU Real Research Group

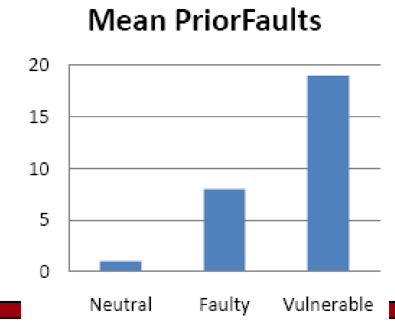
http://www.123rf.com/photo_1268359_needle-in-a-haystack-finding-or-loosing-things.html

Subject Project: Firefox 2.0









Agenda

- Using metrics to predict the presence of security vulnerabilities in code
  - Static analysis alerts (M. Gegick)
  - Developer activity metrics
  - Complexity
  - Traditional code metrics
- Misc observations

Hypotheses: Static Analysis

- Above a statically determined threshold, static analysis vulnerability alerts are in the same components as vulnerabilities that are likely to be exploited.

If a developer has such poor coding practices that he/she causes lots of static analysis alerts, you should look carefully in that area for other implementation bugs and larger design flaws.

Static Analysis Alerts

- **Hypothesis 1**: Source code analysis tool alerts are in the same component as additional coding vulnerabilities and vulnerabilities associated with the design and operation of the software system.
- **Hypothesis 2**. Additional metrics that include code churn and size, churn, coupling, and faults found manually increase the accuracy of a predictive model that uses source code static analysis alerts alone.

Empirical Case Studies on Three Commercial Software Systems

- Three commercial telecommunications software systems
  - Two systems from one anonymous vendor
  - Cisco Systems system
- Each system has over one million source lines of C/C++ code
- Each system is in a different telecommunications product sector.

Correlations between static analysis alerts and vulnerability count are positive and significant.

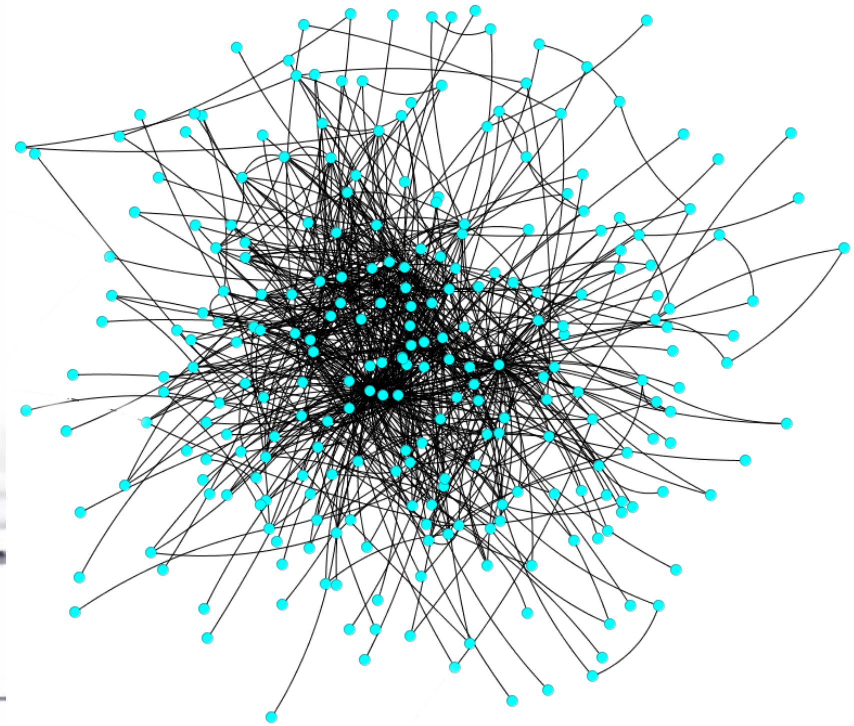
Metric	Case study 1 (component-level)	Case study 2 (file-level)	Case study 2 (component-level)	Case study 3 (component-level)
All SA alerts	0.2	0.2	0.6	0.2
Security SA alerts	0.2	0.2	0.5	0.2

- Since correlations are significant, these metrics can be used in statistical models.
- Security-related alerts have same correlation as all alerts
- Implication – no need to sift through static analysis alerts to use as predictor

Agenda

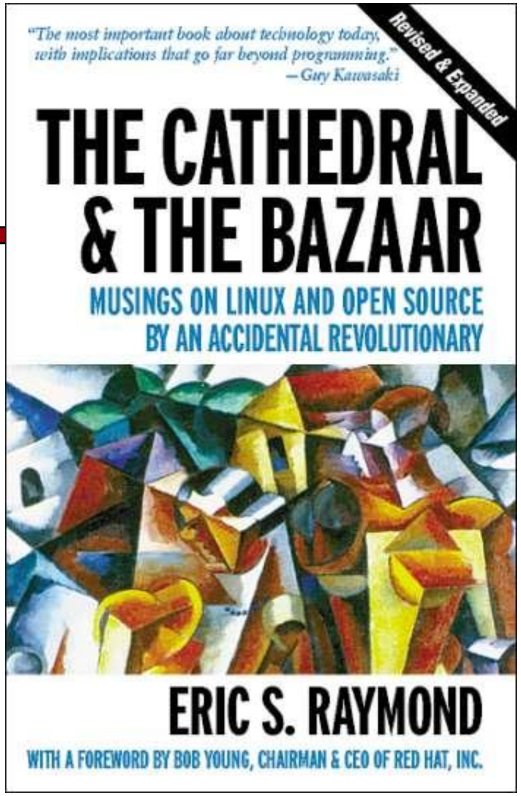
- Finding the security vulnerabilities in code
  - Static analysis alerts
  - Developer activity metrics (A. Meneely)
  - Complexity
  - Traditional code metrics
- Misc observations

Software is about People



Team Problems → Software Problems

Linus’ Law & Security



“*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

More Co-Developers → Diverse perspectives → Large knowledge base → Secure Software

Is this *really* true? **(*Do the numbers match up?*)**

– More people → Too many cooks in the kitchen?

Case Studies

Three empirical case studies

- RHEL4 Linux kernel, PHP, and Wireshark
- Pre-release version control logs
- Post-release security vulnerabilities
- Viewed files as **vulnerable** (>0 vulnerabilities) or **neutral** (none found yet)

	RHEL4 kernel	PHP	Wireshark
Number of committers	557	84	19
Source code files	14,454	1,039	2,688
% files vulnerable	3%	6%	3%
Pre-release version control log data	16 months	2 years	2 years
Years of security data	5 years	3 years, 5 months	3 years, 5 months

How Many Developers?

- Metric: NumDevs

  The number of distinct developers who changed a given source code file

In all three case studies…

Vulnerable files had more developers than neutral files (p<0.001, MWW)

*Files changed by **6 or more developers** were **4 times more likely** to have a vulnerability, (p<0.001, MWW)*

(…not quite what Linus' Law says…)

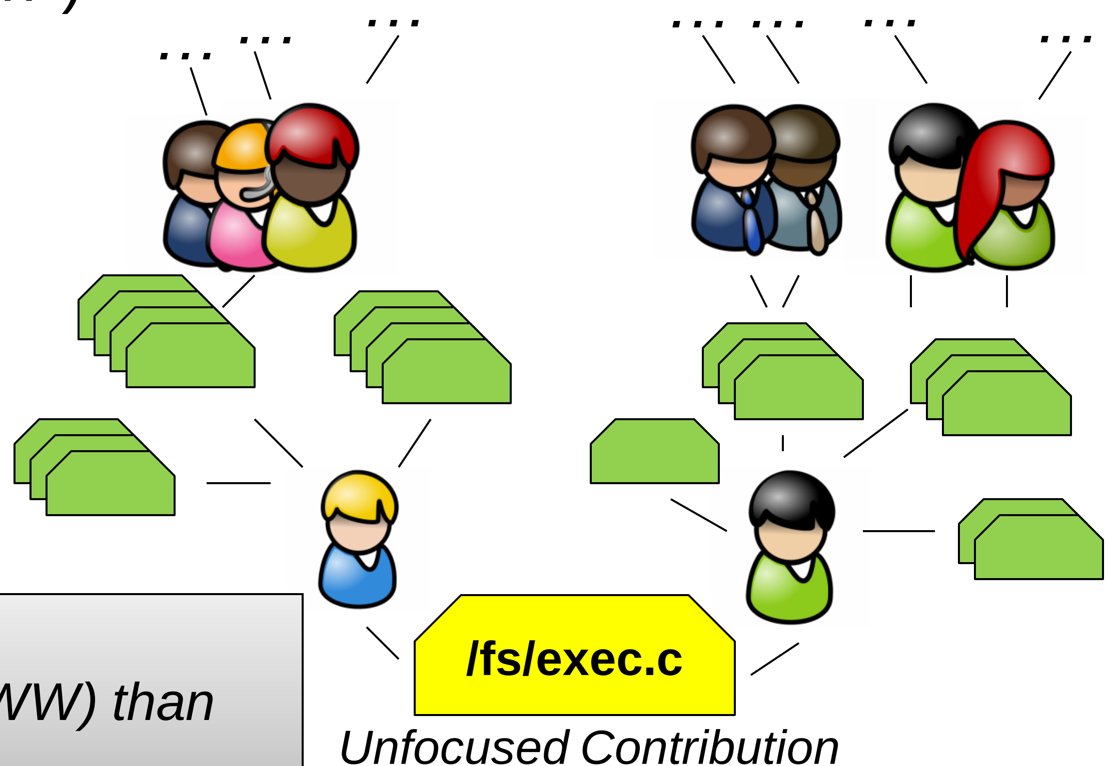
Unfocused Contributions

Examined files changed by **many** developers who were **working on many other files** at the time (an "*unfocused contribution*")

Take into account the other files that the contributing developers were working on

Used contribution network centrality (**CNBetweenness**)

Vulnerable files had a higher CNBetweenness (p<0.001, MWW) than neutral files.



Agenda

- Finding the security vulnerabilities in code
  - Static analysis alerts
  - Developer activity metrics
  - Complexity (Y. Shin)
  - Traditional code metrics
- Misc observations

Why Complexity and Complexity Metrics Matter?

- Security experts say
  - Bruce Schneier
    - “Complexity is the worst enemy of security”
  - Dan Geer
    - “Complexity provides both opportunity and hiding places for attackers”
  - Gary McGraw
    - “A third trend impacting software security is unbridled growth in the size and complexity of modern information systems, especially software systems”
- Complex code is difficult to understand, test, and maintain
- Can complexity metrics find vulnerable code locations?

Subject Projects

- Firefox
  - 34 releases from Release 1.0 to Release 2.0.0.16
  - 11 combined releases consisting of three to four minor releases
- Red Hat Enterprise Linux 4 kernel (RHEL4)

Project	# of Files	LOC	Files with Vulns.	% of Files with Vulns.
Firefox	10,320 ~ 11,080	2 MLOC ~ 2.3 MLOC	14 ~ 123	**0.126% ~ 1.192%**
RHEL4	13,568	3 MLOC	194	**1.4%**

Metrics

- 14 code complexity metrics
  - e.g. lines of code, cyclomatic complexity, comment density
- 3 code churn metrics
  - e.g. Frequency of file changes, lines of code changed, and new lines of code
- 11 developer metrics
  - e.g. Number of developers, betweenness, closeness

Results: Discriminative Power

- Most metrics provided discriminative power at $p < 0.05$

	# of metrics	Firefox	RHEL
Code complexity	14	13	13
Code churn	3	3	3
Developer	11	10	9

Agenda

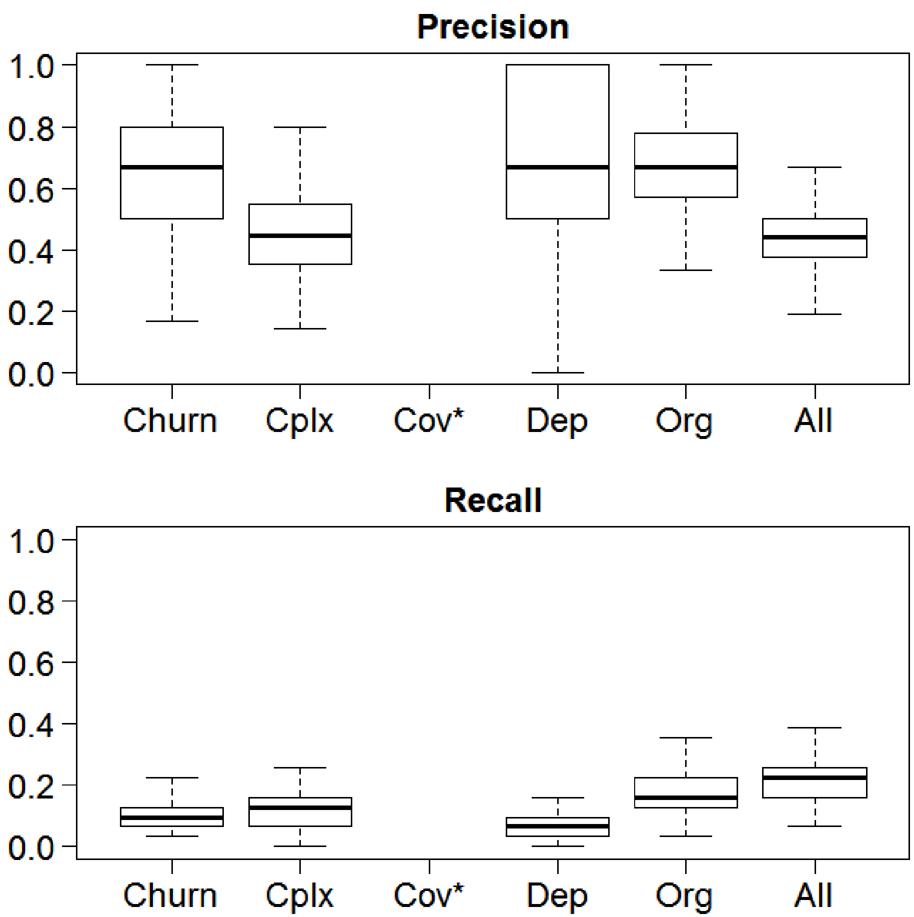
- Finding the security vulnerabilities in code
  - Static analysis alerts
  - Developer activity metrics
  - Complexity
  - Traditional code metrics (Shin, Zimmerman, Gegick, Morrison)
- Misc observations

Support for Traditional Metrics with Windows Vista (Zimmerman)

Metric	rho
Edit Frequency (EF)	0.292
Total Lines of Code	0.281
Frequency	0.279
Total Complexity	0.276
Repeat Frequency	0.273
Number of Ex-Engineers (NOEE)	0.270
TotalFanIn	0.263
TotalFanOut	0.262
Number of Engineers (NOE)	0.261
Total Global Variables	0.255
Total Churn	0.254
Max FanIn	0.224
Max Complexity	0.207
Max FanOut	0.196
Max Lines of Code	0.194
Outgoing direct	0.168
Total ClassMethods	0.167
Max ClassMethods	0.164
Total InheritanceDepth	0.161
Total BlockCoverage	0.157
Incoming direct	0.156
Tota ClassCoupling	0.154
Total ArcCoverage	0.152
Incoming closure	0.148
Total SubClasses	0.141
Max InheritanceDepth	0.137
Max ClassCoupling	0.137
Max SubClasses	0.124
Level of Org. Code Ownership (OCO)	0.123
Depth of Master Ownership (DMO):	0.101

All correlations values are significant at $p<0.0001$.

More on Windows Vista



What you look at will likely be a vulnerability …

…. But many vulnerabilities will be missing.

Agenda

- Finding the security vulnerabilities in code
  - Static analysis alerts
  - Developer activity metrics
  - Complexity
  - Traditional code metrics (Shin, Zimmerman, Gegick, Morrison)
- Misc observations

Comparison of Fault Prediction and Vulnerability Prediction (Shin)

- Goal
  - Investigate whether fault prediction metrics models are equal to or better than vulnerability prediction models in predicting vulnerable code locations when the same traditional fault prediction metrics are used
- Hypothesis
  - A vulnerability prediction model can predict vulnerable code locations better than a fault prediction model
- Metrics
  - Code complexity, code churn, and prior fault history metrics
- Subject project
  - Firefox 2.0 and its minor releases

Observations

- When built with traditional fault prediction metrics, vulnerability prediction performance is similar when the model is trained on all faults and when it is trained on vulnerabilities

Observations - 1

- Static Analysis Alerts
  - Predictive: Static analysis alerts are indicative of all security vulnerabilities.
  - No pre-processing to determine true positive necessary

Observations - 2

- Developer activity metrics
  - Actionable and predictive
    - Don't allow too many people to change same (critical) file
    - Watch for the "hummingbirds" that change many files.
- Complex code
  - Actionable and predictive: Complex code is less secure

Observations - 3

- Traditional code metrics
  - Predictive: Traditional code metrics can be used to find vulnerabilities
  - Support that vulnerabilities have the same characteristics as faults