Predicting Attack-prone Components

Michael Gegick

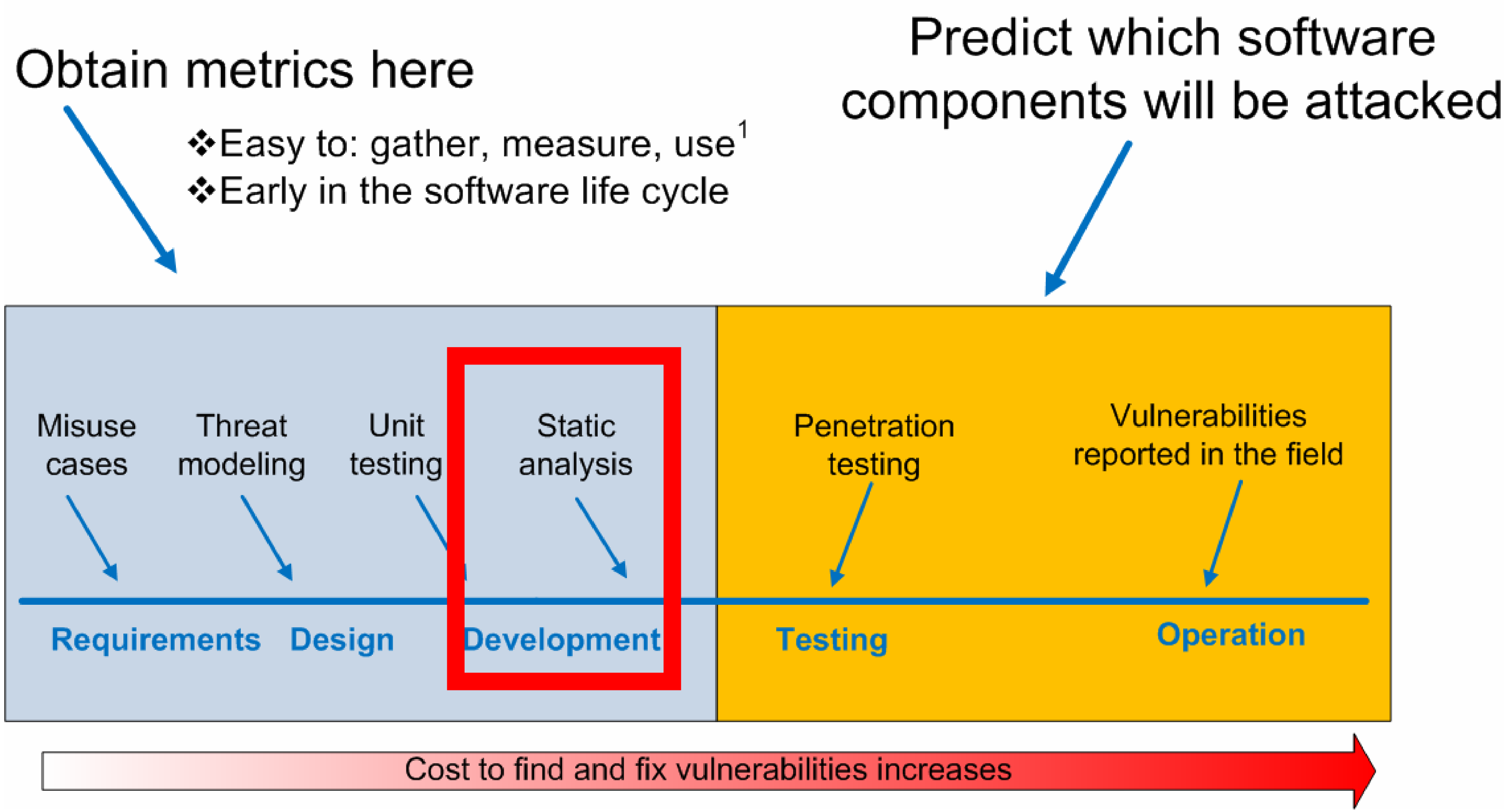
22 May 2009

Security should be designed and built into the software [1]

- Software security: Build security into the software [4]
  - Incorporating security into the software life cycle has reduced count of serious vulnerabilities at Microsoft[1]

[1]http://eweek.com/article2/0,1759,1779769,00.asp

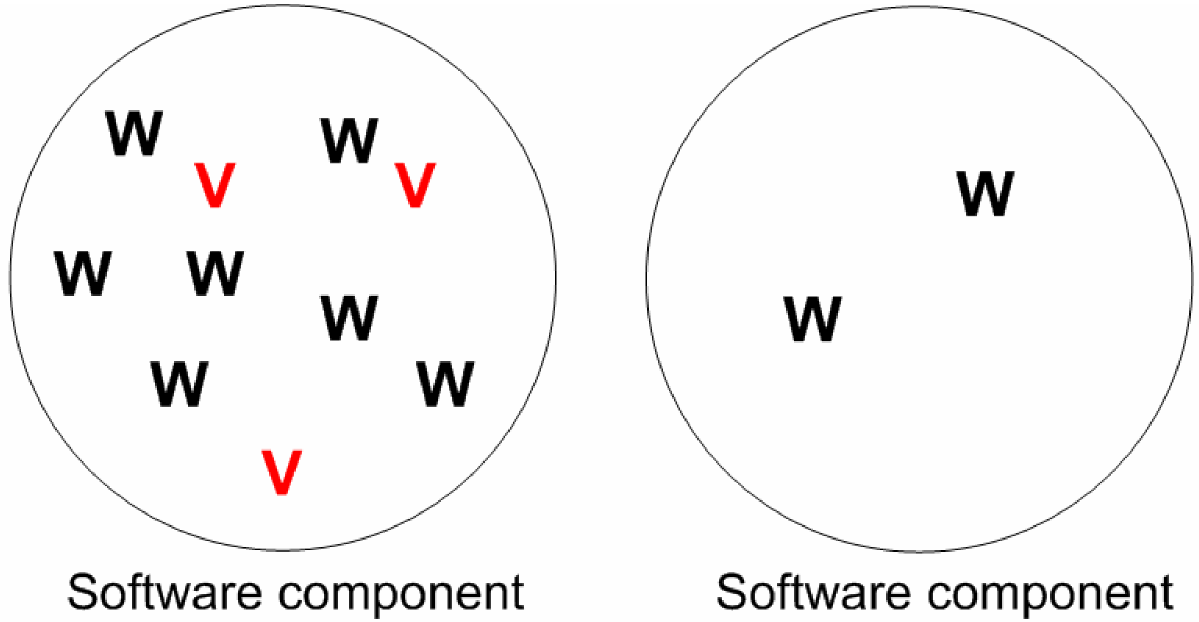
Challenge: The costs to identify faults increases downstream in the software life cycle [2].



The goal of this research is to reduce vulnerabilities from escaping into the field. We incorporate metrics into statistical models that predict which components are most susceptible to attack .

[1]ISO/IEC 24765, "Software and Systems Engineering Vocabulary," 2006.

No single fault detection technique can identify all faults in a software system [5].



W – source code static analysis warning

V – vulnerabilities that are not detectable by source code static analyzers and that will likely be exploited

H_A: above a statistically determined threshold, source code static analysis tool warnings are predictive of other vulnerabilities identified during testing and in the field.

Background: Defining Vulnerability- and Attack-prone Components

Reliability context

Security context

Fault-prone component → **Vulnerability-prone component**

Likely to contain faults → **Likely to contain vulnerabilities**

Failure-prone component → **Attack-prone component**

Likely to cause failures → **Likely to be exploited**

component - "one of the parts that make up a system" [3]

Reliability concepts may be applicable in the security realm.

Research objective: predict which components are attack-prone.

- Attack-prone components[1] are those components that have at least one vulnerability identified during testing or reported by customers or third-party researchers.

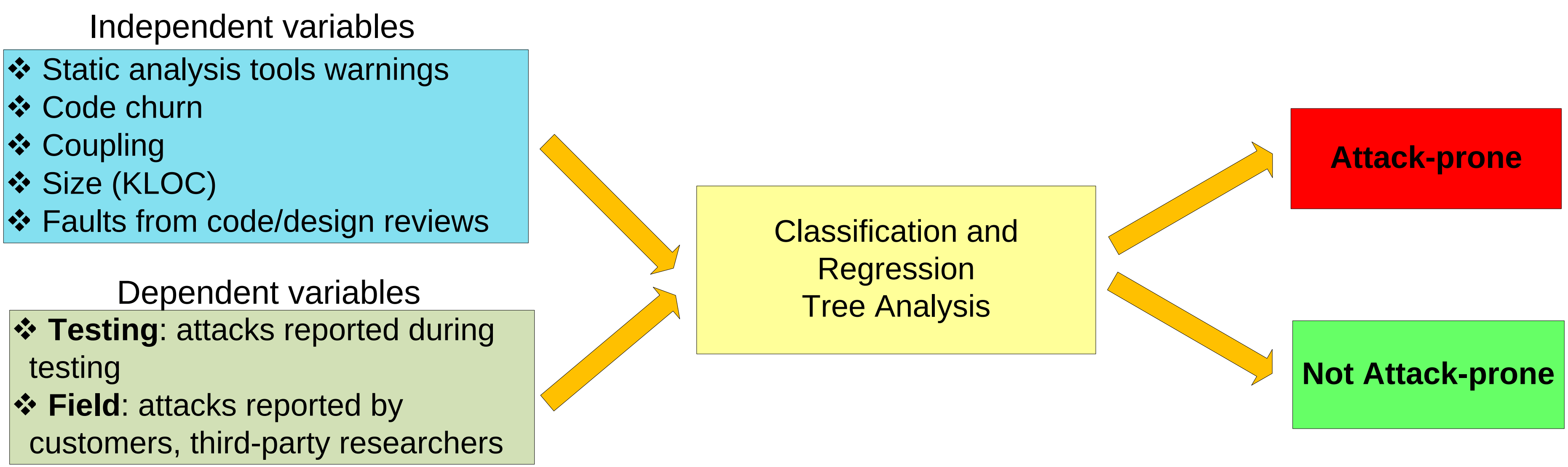
[1]Multiple files per component in the context of this research.

Prioritize security fortification efforts to the attack-prone components.

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.

Classification and Regression Trees (CART) used as statistical approach



Other approaches that were examined, but found to be less effective

- **Logistic regression**
- **Discriminant analysis**
- **Zero-inflated Poisson**
- **Zero-inflated negative binomial**

Threats to Validity

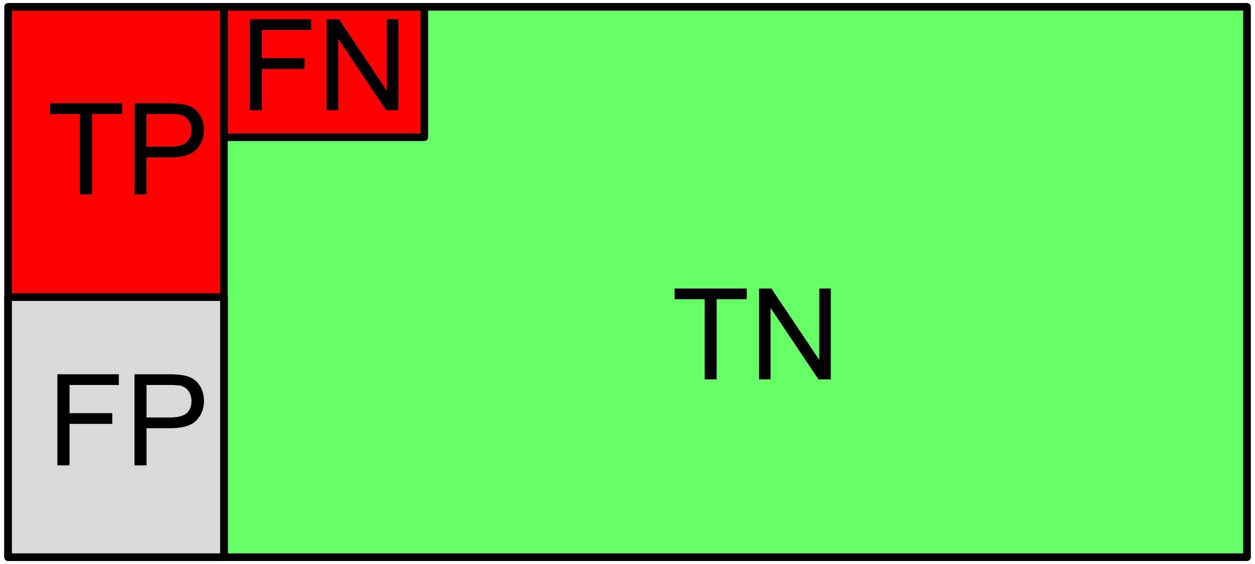
- Residual vulnerabilities in software are possible.
- Vulnerability count is a function of security testing effort and customer usage, where effort and usage are not equal for all components.
- Identified vulnerabilities are scarce. Confidence in statistical results can be low as a result.
- Results are from three software systems. They are not representative for all software systems.

Correlations between metrics 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)
Non-security failures	0.8	0.4	0.7	0.4
Code churn	0.4	0.4	0.7	0.2
Size (SLOC)	0.4	0.4	0.6	0.2
Coupling Metric	N/A	0.2	0.6	N/A
SCSA warnings	0.2	0.2	0.6	0.2
SCSA security warning	0.2	0.2	0.5	0.2

- Since correlations are significant, these metrics are used in statistical models.
- Non-security failure count among the strongest correlations for all metrics and case studies.
- Reliability engineers should look for vulnerabilities in the most failure-prone components.

CART results: Source code metrics can prioritize security fortification efforts to attack-prone components.



Region size proportional to count of components

True Positives (TP) + False Positives (FP): 18.6% of system components
False Positives: 9.1%

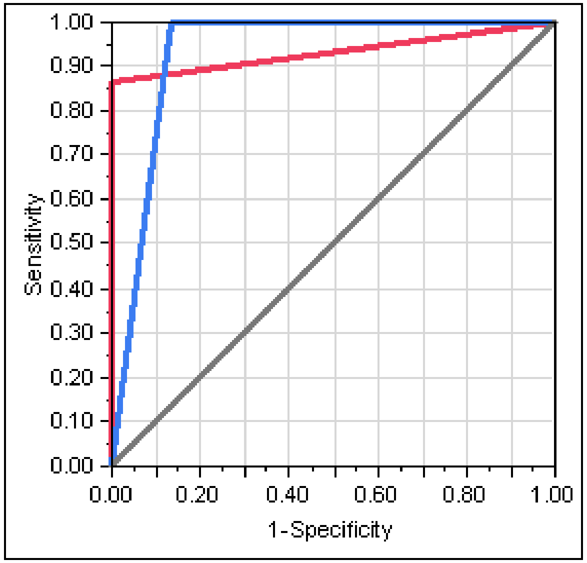
Accuracy: 88.0%
Precision: 52.5%
Recall: 75.6%

Model prioritizes security efforts in TP and FP regions.

TN (True Negatives - correctly classified as not attack-prone)
FN (False Negatives - misclassified as not attack-prone)
TP (True Positives - correctly classified as attack-prone)
FP (False Positives - misclassified as attack-prone)

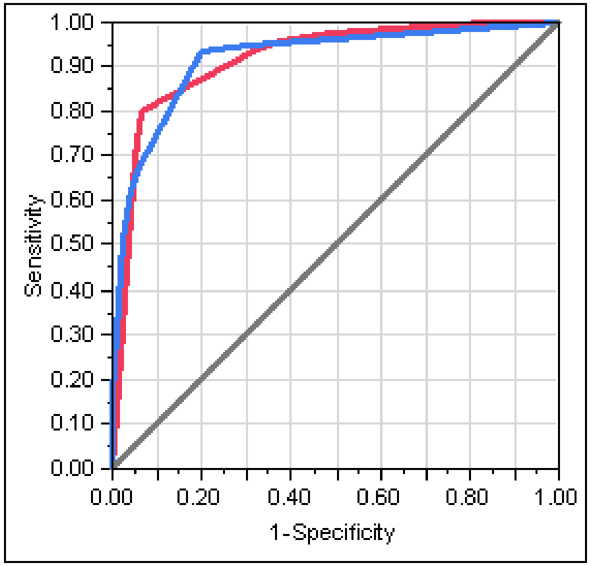
Area under the curve (AUC) is not dissimilar for three case studies

Case study 1



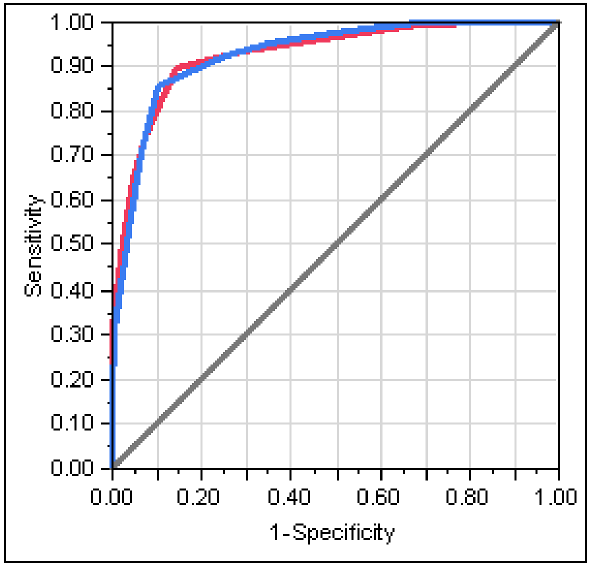
AUC = 93.0%

Case study 2



AUC = 91.9%

Case study 3



AUC = 94.4%

Source code static analysis warnings are an important predictor

G^2 likelihood-ratio chi-square statistic.

	SCSA warnings	Churn	Static inspections	File coupling
Case study 1	10.6	12.2	N/A	N/A
Case study 2	32.2	156.6	N/A	18.6
Case study 3	76.1	24.9	20.2	N/A

- Larger G^2 indicates better fit to the data.

Components with source code static analysis warnings may also have other types of vulnerabilities.

References

[1] Anderson J., "Computer Security Technology Planning Study," Fort Washington, October 1972.
[2] Boehm B., *Software Engineering Economics*, New Jersey, Prentice-Hall, 1981.
[3] IEEE, "ANSI/IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990)," IEEE Computer Society Press, Los Alamitos, CA, 1990.
[4] McGraw G., *Software Security: Building Security In*, Boston, Addison-Wesley, 2006.
[5] Young M. and R. N. Taylor, "Rethinking the Taxonomy of Fault Detection Techniques," *ICSE*, pp. 53-62, 1989.