Quantitative Cyber-Security

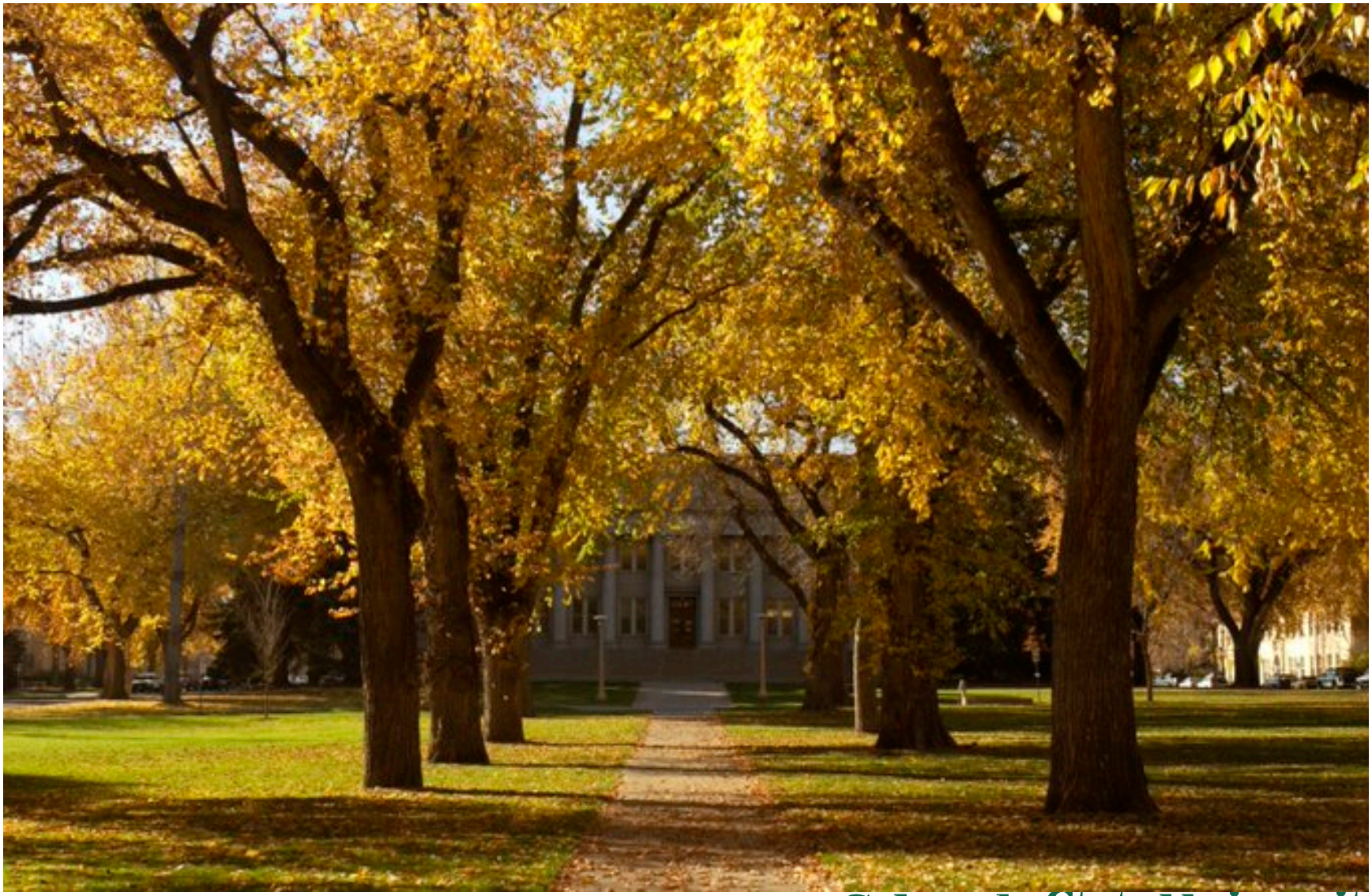
Colorado State University
Yashwant K Malaiya

CS559

L15: CVSS & Testing

CSU Cybersecurity Center
Computer Science Dept

Leaves are falling..



Colorado State University

Notes

Midterm coming Tuesday.

Will use canvas. Will need proper laptop/pc with camera.

- Sec 001: Respondus 3:30-4:45 PM
- Sec 801: Honorlock Time window will be announced later
  - 801 students local in Fort Collins need to take it during 3:30-4:45 PM

Quick review for MT this Thursday.

Some Quiz Questions: Q6

Q. According to a paper by Bilge and Dumitras, Vulnerability lifecycle events are ..

- introduction, discovery, exploit release, disclosure, anti-virus signature available, patch available, patch installed

Q. In-class question on Thursday: Address Space Layout Randomization is an example of

- Proactive Defense

Q. **For this question, data for a certain product is provided in file - Use the data to fit the AML model using Excel Solver. Use the initial values A=0, B=115, C=1. What is the value of A obtained?**

- Between 0.0009 and 0.001

Comment: can be done without using an array formula, however the spreadsheet is cleaner with an array formula.

OpenOffice also has a solver. MATLAB has an Optimization toolbox.

Some Quiz Questions: Q7

Q. **Discovering previously unknown vulnerabilities ...**

- Is legal and can be profitable

Q. The _____ can be used to identify possible seasonality. Identify all correct answers.

- autocorrelation function, seasonal index

Q. CVSS is a system for ..

- assessing severity of the software vulnerabilities.

Q. The occurrence rate of breaches is a _situation _metric

Q. About a third of the vulnerabilities have Low Severity according to CVSS V. 2.0 False

Q. Top 10 songs for each year, and the songs are ranked according to popularity. What kind of scale does this ranking represent?

- Ordinal

CVSS: Review

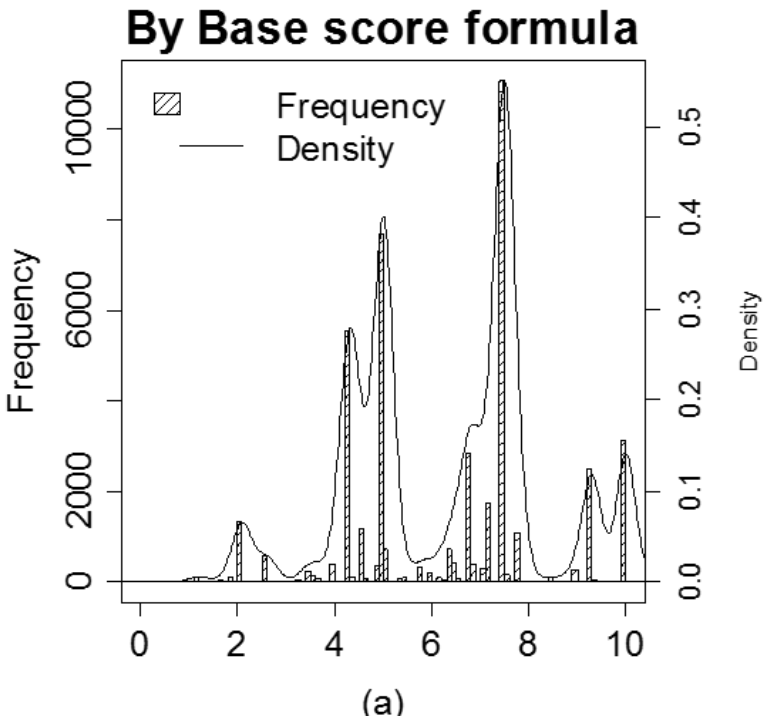
- Developed to assess severity levels of vulnerabilities.
- V3: **L**: 0.1-, **M**:4.0-, **H**: 7.0-, **Crit**: 9.0-10.0

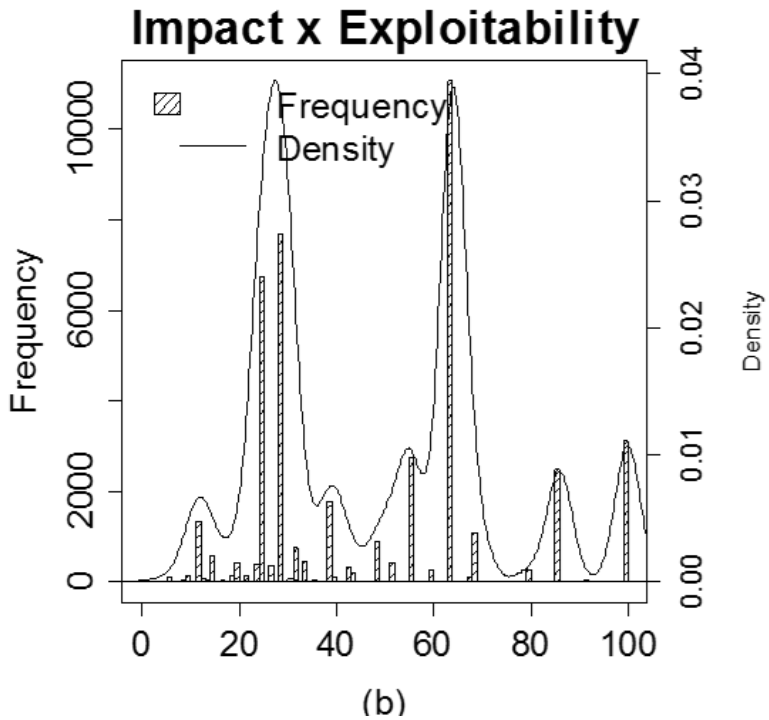
Formulas:

- Base Score = f(Impact, Exploitability)
- Sub-scores Exploitability and Impact are computed using Base Metrics, that depend on the vulnerability itself.
- Temporal Score = f(Base score, exploit, patch)
- Environmental score = f(CIA requirements, developments)

CVSS: How useful it is?

- What if they had multiplied exploitability and impact sub-scores instead of adding?
- Correlation among
  - CVSS Exploitability
  - Microsoft Exploitability metric
  - Presence of actual exploits
- Time to discovery?
- Reward program?
- Can metric/score determination be automated?





	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Combinations
(a)	0	5	6.8	6.341	7.5	10	63
(b)	0	29	49	48.59	64	100	112

H. Joh and Y. K. Malaiya, "Defining and Assessing Quantitative Security Risk Measures Using Vulnerability Lifecycle and CVSS Metrics," SAM'11, The 2011 International Conference on Security and Management, pp.10-16, 2011

Colorado State University

Has CVSS worked?

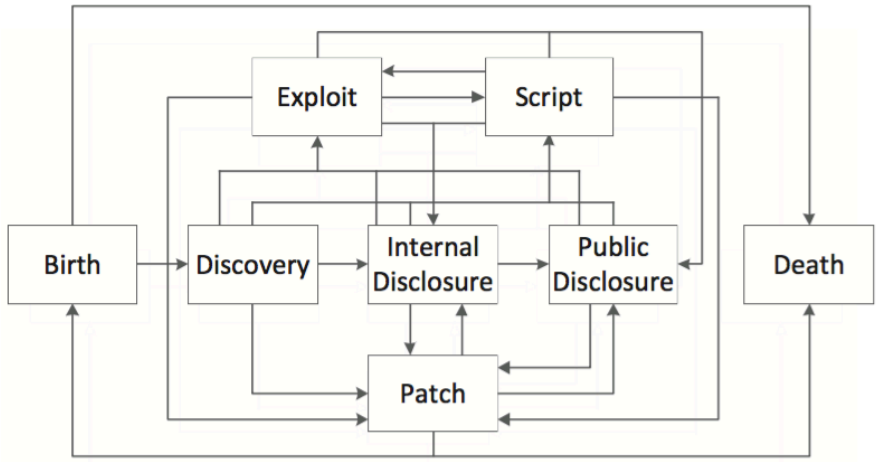
- Windows 7 Correlation among
  - CVSS Exploitability
  - Microsoft Exploitability metric
  - Presence of actual exploits
- No significant correlation found.
- Continuing research

Variables	Exploit Existence	MS-EXP	CVSS-EXP
Exploit Existence	1	-0.078	-0.146
MS-EXP	**-0.078**	1	-0.116
CVSS-EXP	**-0.146**	-0.116	1

A. Younis and Y.K. Malaiya, "Comparing and Evaluating CVSS Base Metrics and Microsoft Rating System", The 2015 IEEE Int. Conf. on Software Quality, Reliability and Security, pp. 252-261

Likelihood of Individual Vulnerabilities Discovery

- **Ease of discovery**
  - Human factor (skills, time, effort, etc.), Discovery technique, Time
- Time:



- Apache HTTP server
- CVE-2012-0031, **(01/18/2012)**
- V. 1.3.0→1998-06-06

Time to Discovery = Discovery Time Date – First Effected version Release Date

Colorado State University

10

Correlation: Access Complexity vs Time to Discover

❖ Access complexity vs Time to Discover

- **AC= Low**

Min.	1st Qu.	**Median**	Mean	3rd Qu.	Max.
0.100	0.900	**2.000**	3.338	4.500	18.000

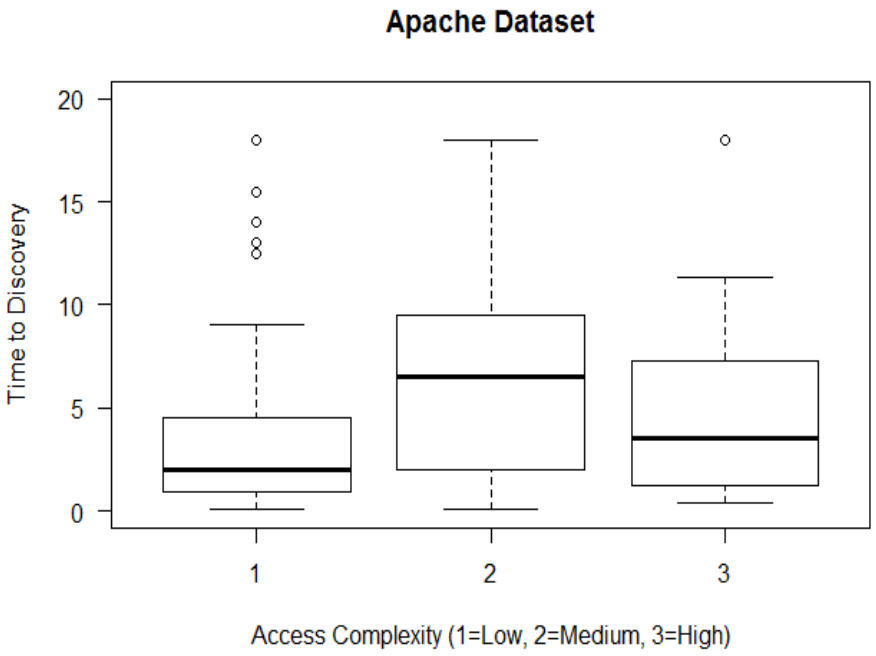
- **AC= Medium**

Min.	1st Qu.	**Median**	Mean	3rd Qu.	Max.
0.100	2.000	**6.500**	6.819	9.500	18.000

- **AC= High (very few points)**

Min.	1st Qu.	**Median**	Mean	3rd Qu.	Max.
0.400	1.350	**3.500**	5.208	7.125	18.000

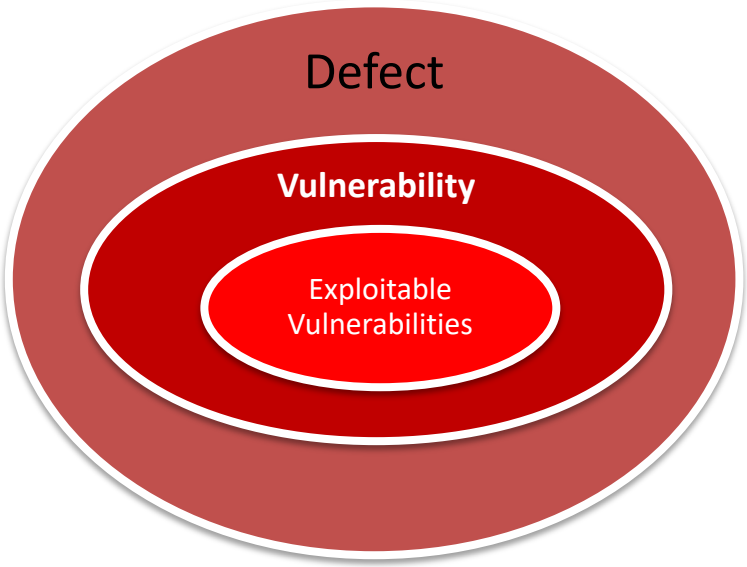
- **There may be come correlation between Acces Complexity and Time to Discover**



11

Characterizing Vulnerability with Exploits

- **1 to 5 %** of defects are vulnerabilities.
- Finding vulnerabilities can take considerable expertise and effort.
- Out of 49599 vulnerabilities reported by NVD, **2.10% have an exploit**.
- A vulnerability with an exploit written for it presents more risk.
- **What characterizes a vulnerability having an exploit?**



Vulnerability	In-Degree	Out-Degree	CountPath	ND	CYC	Fan-In	No of Invocation	SLOC	Exploit Existence
CVE-2009-1891	1	9	9000	6	68	45	2	211	NEE
CVE-2010-0010	4	9	145	4	11	16	4	38	EE
CVE-2013-1896	26	5	8	1	5	37	3	29	EE

Colorado State University

12

CVSS Base Score vs Vulnerability Rewards Programs

- We examined 1559 vulnerabilities of Mozilla Firefox and Google Chrome browsers for which records were available.
- Looked at Mozilla and Google vulnerability reward programs (VRPs) records for those vulnerabilities.

Firefox		
Vulnerabilities	Rewarded	Not rewarded
547	225	322
VRP severity	Rewarded	Not rewarded
Critical & High	210	202
Medium	15	89
Low	0	31

A. Younis, Y. Malaiya and I. Ray, "Evaluating CVSS Base Score Using Vulnerability Rewards Programs", Proc. 31th Int. Information Security and Privacy Conference, IFIP SEC, Ghent, Belgium, 2016, pp. 62-75.

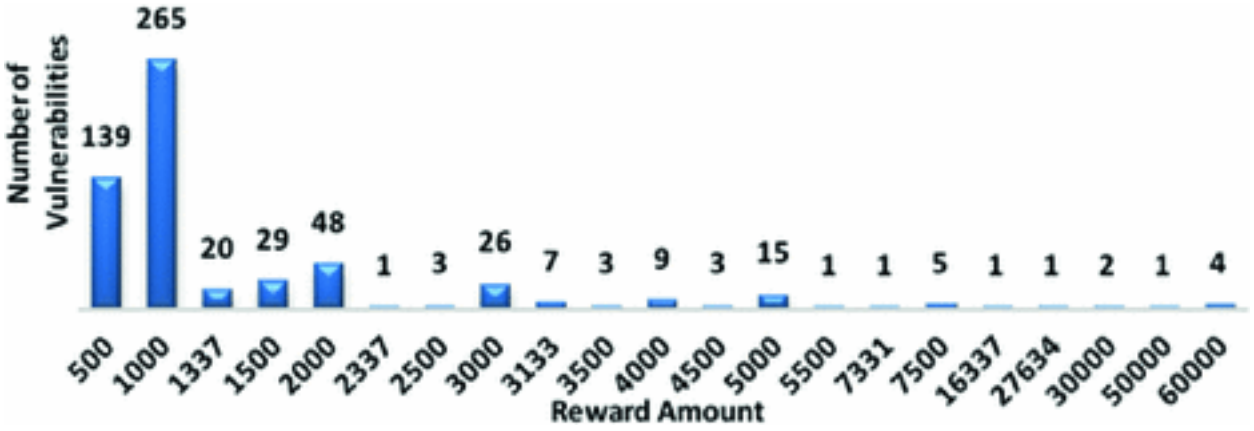
CVSS Base Score vs Vulnerability Rewards Programs

Chrome		
Vulnerabilities	Rewarded	Not rewarded
1012	584	428
VRP severity	Rewarded	Not rewarded
Critical & High	441	175
Medium	136	137
Low	7	116

The results results show that CVSS Base Score may have some correlation with the vulnerability reward program.

How much did Chrome pay?

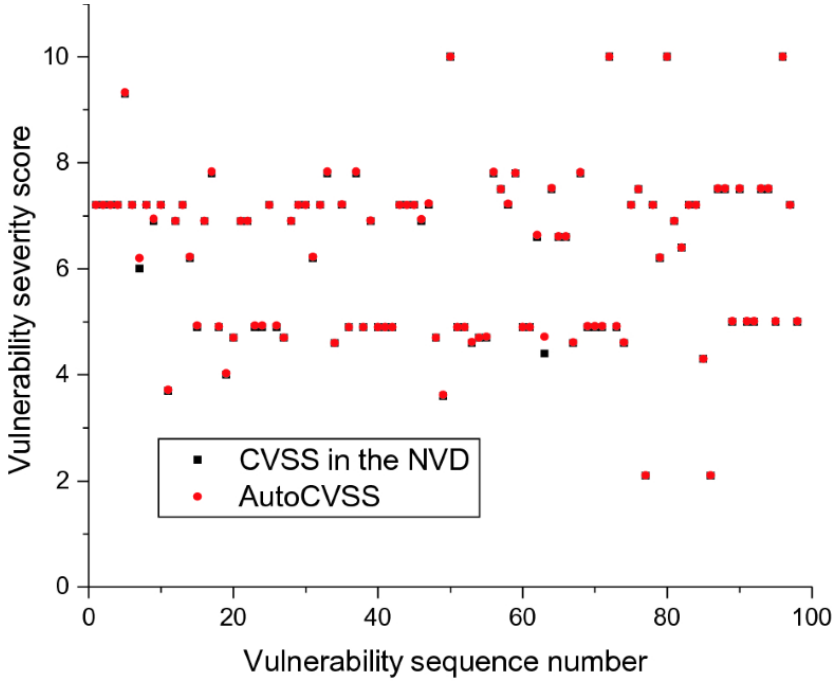
- Incidental result



Colorado State University

AutoCVSS?

- Zou et al. 2019: 98 vulnerabilities from Linux kernel, FTP service, and Apache service with their exploits from exploit-db.
- CVSS relies on human experts to determine metric values during the process of vulnerability severity assessment. They have attempted to automate the process.
- Result is that only two vulnerability severity scores assessed by AutoCVSS are obviously different from those in the NVD for CVSS v2.



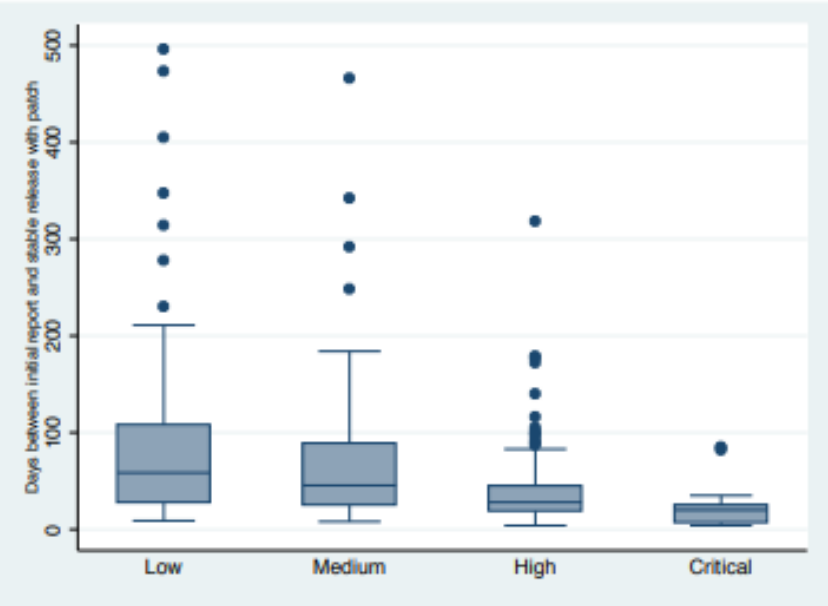
D. Zou, J. Yang, Z. Li, X. Ma, , AutoCVSS: An Approach for Automatic Assessment of Vulnerability Severity Based on Attack Process, Int. Conf. on Green, Pervasive, and Cloud Computing, April 2019

VRP Cost effectiveness

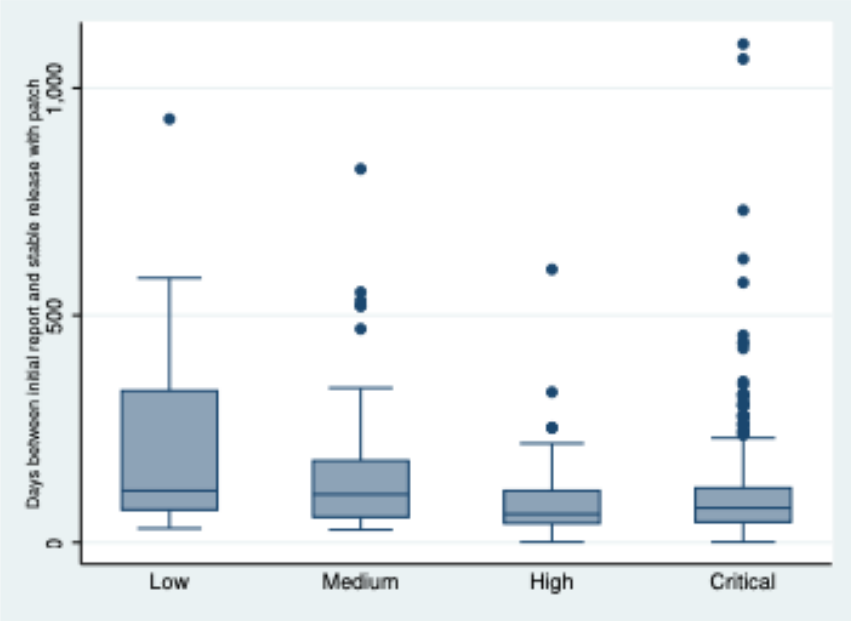
- Hypothesis: A VRP can be a cost-effective mechanism for finding security vulnerabilities.
  - Period studied: 7/09-1/13
  - Chrome's VRP has cost $485 per day on average, and that of Firefox has cost $658 per day.
  - Average North American developer on a browser security team (i.e., that of Chrome or Firefox) would cost around $500 per day (assuming a $100,000 salary with a 50% overhead).
- Hypothesis: Contributing to a single VRP is, in general, not a viable full-time job, though contributing to multiple VRPs may be, especially for unusually successful vulnerability researchers.
- Hypothesis: Successful independent security researchers bubble to the top, where a full-time job awaits them

M. Finifter, D. Akhawe, D. Wagner, An empirical study of vulnerability rewards programs. In USENIX Security Symposium 2013 (2013), pp. 273-288

Time to patch



(a) Chrome



(b) Firefox

M. Finifter, D. Akhawe, D. Wagner, An empirical study of vulnerability rewards programs. In USENIX Security Symposium 2013 (2013), pp. 273-288

Quantitative Security

Colorado State University

Yashwant K Malaiya

CS 559

Testing

CSU CyberCenter
***Course Funding Program* – 2019**

Faults

- Faults cause a system to respond in a way different from expected.
- Faults can be associated with bugs in the system/software structure or functionality.
  - Structure: viewed as an interconnection of components like statements, blocks, functions, modules.
  - Functionality: Described by the input/output/state behavior, described externally.
  - Both structure and functionality can be described at a higher level and a lower (finer) level.
- Example: a file > classes > methods etc. > statements

21

Testing

- Testing and debugging is an essential part of software development and maintenance. 15-75% cost
  - Static analysis: code inspection
  - Dynamic: involves execution
- Defects cause functionality/reliability and security problem.
- Vulnerabilities are a subset of the defects (1-5%)
  - If exploited, allow violation of security related assumptions.
  - Vulnerability discovery can involve testing with
    - Random tests (Fuzzing)
    - Generated tests base on security requirements
- The following discussion is general for all defects.

Partitioning

- Software can be partitioned to ensure that the software is thoroughly exercised during testing
- It is necessary to partition it to identify tests that would be effective for detecting the defects in different sections of the code.
- For testing purposes, a program may be partitioned either functionally or structurally.
- *Functional partitioning* refers to partitioning the input space of a program.
  - For example, if a program performs five separate operations, its input space can be partitioned into five partitions.
  - Functional partitioning only requires the knowledge of the functional description of the program, the actual implementation of the code is not required.
- *Structural partitioning* requires the knowledge of the structure at the code level.
  - If a software is composed of ten modules (which may be classes, functions or other types of units), it can be thought of as having ten partitions

Y. K. Malaiya, "Assessing Software Reliability Enhancement Achievable through Testing", Recent Advancements in Software Reliability Assurance 2019, pp. 107-138

Sub-Partitioning

- A partition of either type can be subdivided into lower level partitions, which may themselves be further partitionable at a lower level if higher resolution is needed (Elbaum 2001).
- Let us assume that a partition p_i can be subdivided into sub-partitions $\{p_{i1}, p_{i2} \ldots p_{in}\}$.
  - Random testing within the partition p_i will randomly select from $\{p_{i1}, p_{i2} \ldots p_{in}\}$. It is possible that some of them will get selected more often in a non-optimal manner.
  - Code within a sub-partition may be correlated relative to the probability of exercising some faults. Thus the effectiveness of testing may be diluted if the same sub-partition frequently gets chosen.
  - Sub-partitioning has a practical disadvantage when the *operational profile* is constructed, it will require estimating the operational probabilities of the associated sub-partitions.

Testing

- We assume that tests are applied at the inputs and the response is observed at the outputs of the unit-under-test.
- A test detects the presence of a fault(s), if the output is different from the expected output.
- Two test approaches:
  - Functional (or Black-box): uses only the functional description of the unit, not its structure to obtain tests. Often random ("fuzzing")
  - Structural testing: uses the structural information to generate tests. Requires more effort, but can be more thorough.
  - Combined

25

Random Testing

- Termed Black-box, fuzzing when used for vulnerabilities
- Random testing is a form of functional testing. In random testing, each test is chosen such that it does not depend on past tests.
- In actual practice, the "random" tests are generated using Pseudo-random algorithms that approximate randomness.
- As we will discuss later, random testing can be effective for moderate degree of testing, but not for thorough testing.

26

Test coverage

- A single test typically covers (i.e. tests for related faults) several sub-partitions (elements such as functions, branches, statements)
- The coverage obtained by a test-set can be obtained using coverage tools.
- The test coverage achieved by a test-set is given by ratio:

$$\text{coverage} = \frac{\text{Number of elements covered}}{\text{Total number of elements}}$$

Input mix: Test Profile

- The inputs to a system can represent different types of operations. The input mix called "Profile" can impact effectiveness of testing.
- Example:
  - elements e_1, e_2, …e_i, …e_n exercised with probabilities p_1, p_2, …p_i, …p_n
  - Profile then is $\{(e_i, p_i)\}$ for all elements
- For example a Search program can be tested for text data, numerical data, data already sorted etc. If most testing is done using numerical data, more bugs related to text data may remain unfound.

Input mix: Test Profile

- **The ideal Profile (input mix) will depend on the objective**
  - A. Find bugs fast? or
  - B. Estimate operational failure rate?

A. Best mix for **functional** bug finding (Li & Malaiya' 94)

- Quick & limited testing: *Use **operational profile: how the inputs are encountered in actual operation.***
- High reliability: *Probe input space evenly*
  - Operational profile will not execute rare and special cases, the main cause of failures in highly reliable systems.
- Very high reliability: corner cases and rare combinations

B. For **security bugs**: corner cases and rare combinations

- Vulnerability finders / exploiters look for these.

N. Li and Y.K. Malaiya, On Input Profile Selection for Software Testing, Proc. Int. Symp. Software Reliability Engineering, Nov. 1994, pp. 196-205.

H. Hecht, P. Crane, Rare conditions and their effect on software failures, Proc. Annual Reliability and Maintainability Symposium, 1994, pp. 334-337

When you finally catch the person that's been writing bad code all the time

Modeling Bug Finding Process

- The number of bugs found depend on the effort (measured by testing time) and directedness of testing.
- Directedness: looking for bugs
  - In elements not yet exercised enough
    - These will include corner cases
  - Where bugs of a specific type (specially vulnerabilities) are likely to be present.
    - Experience, expertise, intuition

Nature of faults: Detectability Profile

- All faults are not alike.
- There is no such thing as an *average* fault.
- As testing progresses, the remaining faults are the ones harder to find.

Detection Probability

- Detection probability of a fault: if there are N distinct possible input vectors, and if a fault is detected by k of them, then its *detection probability* is k/N.
- A fault with detection probability (dp) 1/N would be hardest to test, since it is tested by only one specific test and none other.
- A fault which is detected by almost all vectors, would have a detection probability close to 1 and will be found with minimal texting effort. It is a low hanging fruit.

Detectability Profile of a unit under test

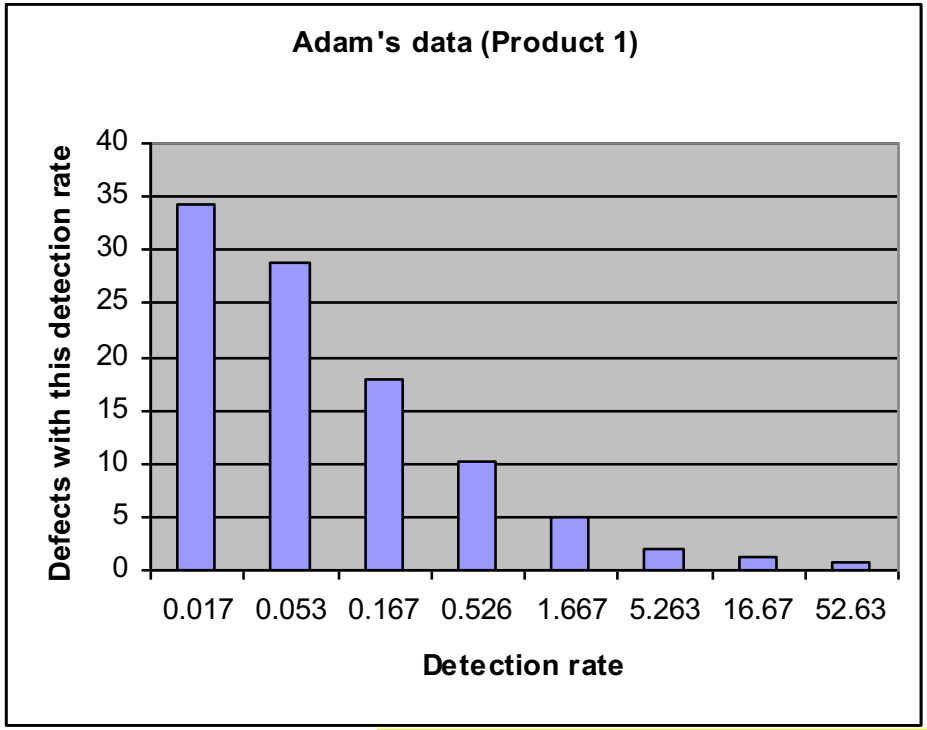
- The Detectability Profile of a unit under test describes how the defects are distributed relative to their detectability.
- Total M faults, total N possible input combinations. The set of faults can be partitioned into these subsets:
- $H = \{h_1, h_2, \ldots h_N\}$
- Where h_k is the number of faults detectable by exactly k inputs. The vector H describes the detectability profile.
  - h_1 is the number of faults that are hardest to find.
  - As testing and debugging continues, harder to find faults will tend to remain. Easy to find faults will get eliminated soon.

Detectability Profile: software

- Regardless of initial profile, after some initial testing, the profile will become asymmetric.
- In the early development phases, inspection and early testing are likely to remove most easy to test bugs, while leaving almost all hardest to test bugs still in.

Detectability Profile: software

- Adam's Data for a large IBM software product. Note bugs with high detection rates are mostly gone.



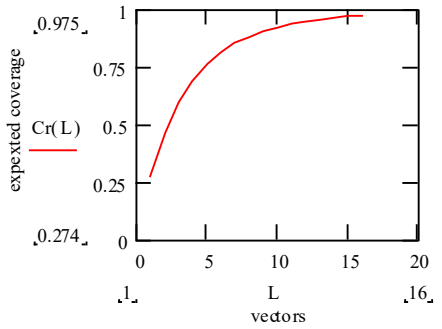
Adams, IBM Journal of Research and Development, Jan. 1984

Coverage Obtained by L Vectors

What fault coverage is achieved by applying L test vector?

- h_k out of M defects detectable by exactly k vectors: detection probability k/N
- P{a defect with dp k/N not detected by a vector} = $(1-\frac{k}{N})$
- P{a defect with dp k/N not detected by L vectors} = $(1-\frac{k}{N})^L$
- Of h_k faults, expected number not covered is $(1-\frac{k}{N})^L h_k$
- Expected test coverage with L vectors

$$C(L) = 1 - \sum_{k=1}^{N} (1-\frac{k}{N})^L \frac{h_k}{M}$$



Y.K. Malaiya and S. Yang, ""The Coverage Problem for Random Testing" Proc. International Test Conference, October 1984, pp. 237-245.

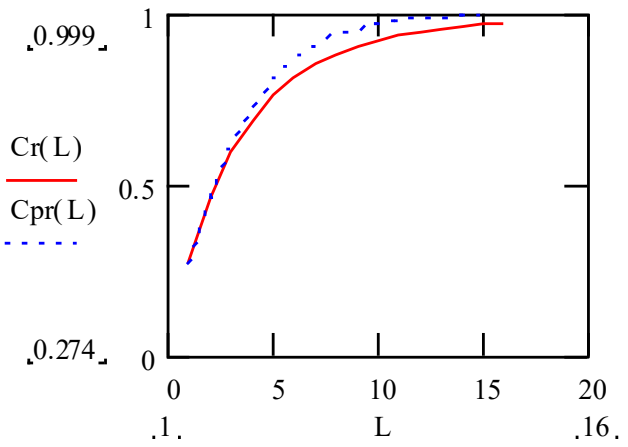
Coverage Obtained by L Vectors

Pseudorandom (PR) testing: a vector cannot repeat, unlike in true Random testing.

- For PR tests (McClusky 87)

$$C(L) = 1 - \sum_{k=1}^{N-L} \frac{{}^{N-L}C_k}{{}^{N}C_k} \frac{h_k}{M}$$

$$\approx 1 - \sum_{k=1}^{N} (1 - \frac{k}{N})^L \frac{h_k}{M} \text{ (for Random)}$$



- For large L, terms with only low k (i.e. faults that are hard to test) have an impact. Thus only lower elements of H need to be estimated.
- For CECL Full Adder,

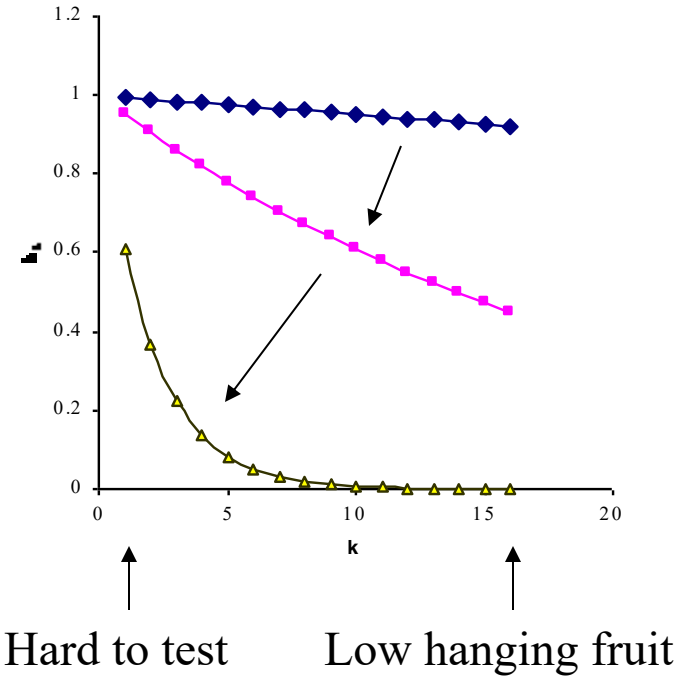
$$C(15) = 1 - [4.2 + 16.4 + 0.9 + 6.3 + 0.84 + 0.03 + 0 + ...] \cdot 10^{-3}$$

K. Wagnor, C. Chin, and E. McCluskey, "Pseudorandom testing. IEEE Trans. Computer, Mar. 1987, pp. 332—343.

Detectability Profile: Software

- Software detectability profile is exponential
- Justification: Early testing will find & remove easy-to-test faults.
- Testing methods need to focus on hard-to-find faults.

As testing time progresses, more of the faults are clustered to the left.



Directed testing

Testing may be directed rather than random because

- Tester may wish to focus on functionality not adequately exercised by random testing (for example recovery code)
- Tester may wish to focus on more critical sections of the code.
- The probability of detecting a fault can be give by p_i, where p_i may be *greater or less than* k/N.

  P{a defect with dp p_i not detected by L vectors} = $(1 - p_i)^L$

- Where $p_i > \frac{k}{N}$ if the previous tests are not repeated, or the test has a good idea of where to look.
- When the *exhaustive set* (ES) of inputs are applied, then P{a defect with dp p_i not detected by ES} ≈ 0
  - Unlikely in most real situations.

Some common models

- Several models for ordinary bug finding process. Termed **Software Reliability Growth Models** (SRGMs).
- **Exponential SRGM**: assumes bug finding rate $\lambda(t)$ is proportional to remaining bugs at time N(t).

$$\lambda(t) = -\frac{dN(t)}{dt} = \beta_1 N(t)$$

- Which has the solution

$$\lambda(t) = \beta_0 \beta_1 e^{-\beta_1 t}$$

- Where β_0 and β_1 are parameters to be determined. B_0 represents the initial number of bugs and β_1 a measure of test effectiveness.

Defect Density

- Exponential defect finding model is
$$\lambda(t) = \beta_0\beta_1 e^{-\beta_1 t}$$
- β_0 represents the initial number of bugs.
- If the initial *defect density* is D(0), and the software size (measured in 1000 lines of code, i.e. KLOC) is S, then
$$\beta_0 = D(0)\times S$$
- The initial defect density is a function of the software development process and the degree of prior defect removal.
- The defect finding rate gradually declines, it takes infinite time to find them all according to the exponential model.
- The final defect density is sometimes used as a release criterion.

SRGM : "Logarithmic Poisson"

- If testing combines random and directed testing, the Logarithmic Poisson arises.
- **Logarithmic Poisson** model, by Musa-Okumoto, has been found to have a good predictive capability

$$\mu(t) = \beta_o \ln(1 + \beta_1 t) \qquad \lambda(t) = \frac{\beta_o \beta_1}{1 + \beta_1 t}$$

- Applicable as long as $\mu(t) \leq N(0)$. Practically always satisfied.
- Parameters β_o and β_1 don't have a simple interpretation. An interpretation has been given by Malaiya and Denton (What Do the Software Reliability Growth Model Parameters Represent?).

43

References

- Y. K. Malaiya, S. Yang, “The Coverage Problem for Random Testing,” IEEE International Test Conference 1984, pp. 237-245.
- Y.K. Malaiya, A. von Mayrhauser and P. Srimani, “An Examination of Fault Exposure Ratio,” IEEE Trans. Software Engineering, Nov. 1993, pp. 1087-1094.
- S. C. Seth, V. D. Agrawal, H. Farhat, "A Statistical Theory of Digital Circuit Testability," IEEE Trans. Computers, 1990, pp. 582-586.
- K. Wagnor, C. Chin, and E. McCluskey, “Pseudorandom testing. IEEE Trans. Computer, Mar. 1987, pp. 332—343.
- E. N. Adams, "Optimizing Preventive Service of Software Products," in IBM Journal of Research and Development, vol. 28, no. 1, pp. 2-14, Jan. 1984.
- J R Dunham, "Experiments in software reliability: Life-critical applications," IEEE Tran. SE, January 1986, pp. 110 - 123
- H. Hashempour, F.J. Meyer, F. Lombardi,, "Analysis and measurement of fault coverage in a combined ATE and BIST environment," Instrumentation and Measurement, IEEE Transactions on , vol.53, no.2, pp.300,307, April 2004.
- Y. K. Malaiya, "Assessing Software Reliability Enhancement Achievable through Testing", Recent Advancements in Software Reliability Assurance 2019, pp. 107-138