Quantitative Security

Colorado State University

Yashwant K Malaiya

CS 559

Vulnerability Discovery Models

CSU Cybersecurity Center
Computer Science Dept

Modeling Vulnerability Discovery

- Quantitative Vulnerability Assessment Alhazmi 2004-2008
- Seasonality in Vulnerability Discovery Joh 2008,2009
- Discovery in Multi-Version Software Kim 2006,2007

Vulnerabilities

Motivation

- For defects: Reliability modeling and SRGMs have been around for decades.
- Assuming that vulnerabilities are special faults will lead us to this question:
  - To what degree reliability terms and models are applicable to vulnerabilities and security? [Littlewood et al].
  - The need for quantitative measurements and estimation is becoming more crucial.

Goal: Modeling Vulnerability Discovery

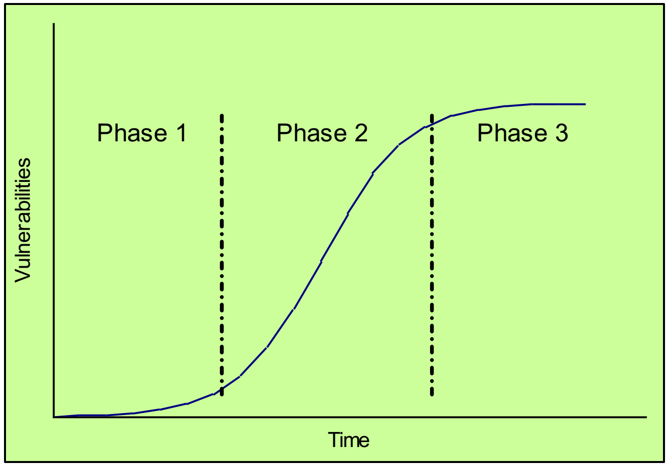
- Developing a quantitative model to estimate vulnerability discovery.
    - Using *calendar time*.
    - Using *equivalent effort*.
- Validate these measurements and models.
    - Testing the models using available data
- Identify security Assessment metrics
    - *Vulnerability density*
    - *Vulnerability to Total defect ratio*

Time – vulnerability discovery model

- What factors impact the discovery process?
  - The changing environment
    - The share of installed base.
    - Global internet users.
  - Discovery effort
    - Discoverers: Developer, White hats or black hats.
    - Discovery effort is proportional to the installed base over time.
    - Vulnerability finders' reward: greater rewards, higher motivation.
  - Security level desired for the system
    - Server or client

Time – vulnerability discovery model

- Each vulnerability is recorded.
  - Available [NVD, vender etc].
  - Needs compilation and filtering.
- Data show three phases for an OS.
- Assumptions:
  - The discovery is driven by the rewards factor.
  - Influenced by the change of market share.



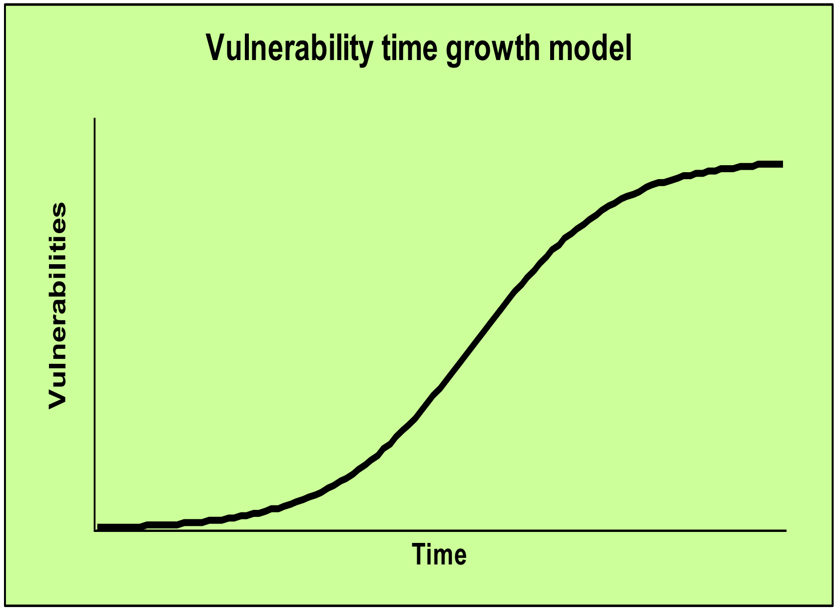
Time–vulnerability Discovery model

$$\frac{dy}{dt} = Ay(B - y)$$

$$y = \frac{B}{BCe^{-ABt} + 1}$$

3 phase model S-shaped model.

- Phase 1:
  - Installed base –low.
- Phase 2:
  - Installed base–higher and growing/stable.
- Phase 3:
  - Installed base–dropping.



Colorado State University

AML Discovery model

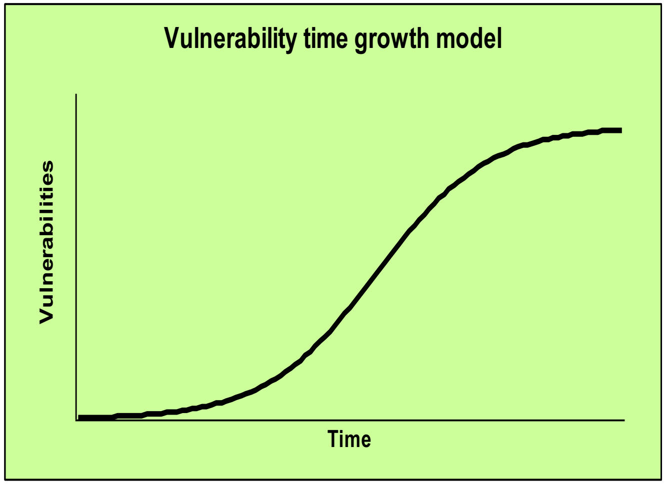
Alhazmi Malaiya Logistic model (AML)

$$\frac{d\Omega}{dt} = A\Omega(B - \Omega), \qquad (3)$$

where Ω is the cumulative number of vulnerabilities, t is the calendar time, and initially t=0. A and B are empirical constants determined from the recorded data.

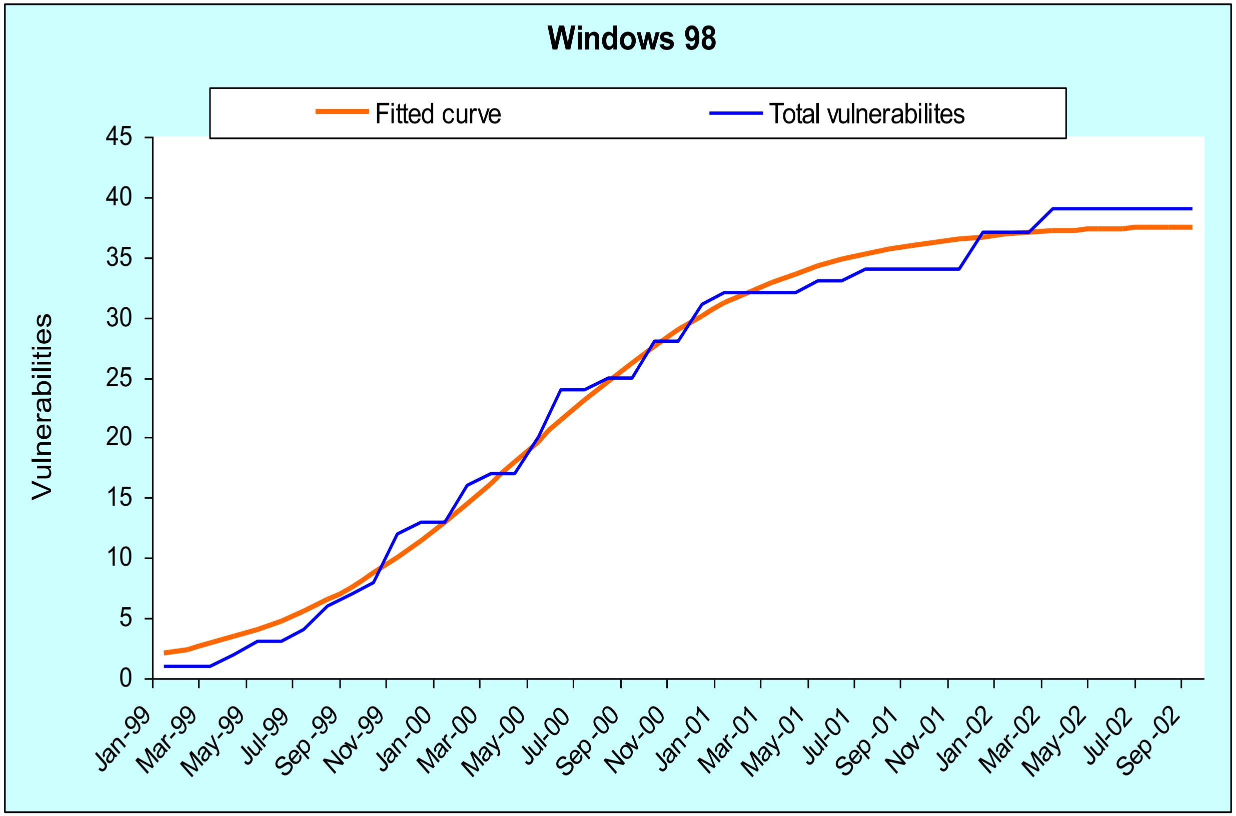
By solving the differential equation we obtain

$$\Omega(t) = \frac{B}{BCe^{-ABt} + 1}, \qquad (4)$$



O. H. Alhazmi and Y. K. Malaiya, "Quantitative Vulnerability Assessment of Systems Software
Proc. Ann. IEEE Reliability and Maintainability Symp., 2005, pp. 615-620

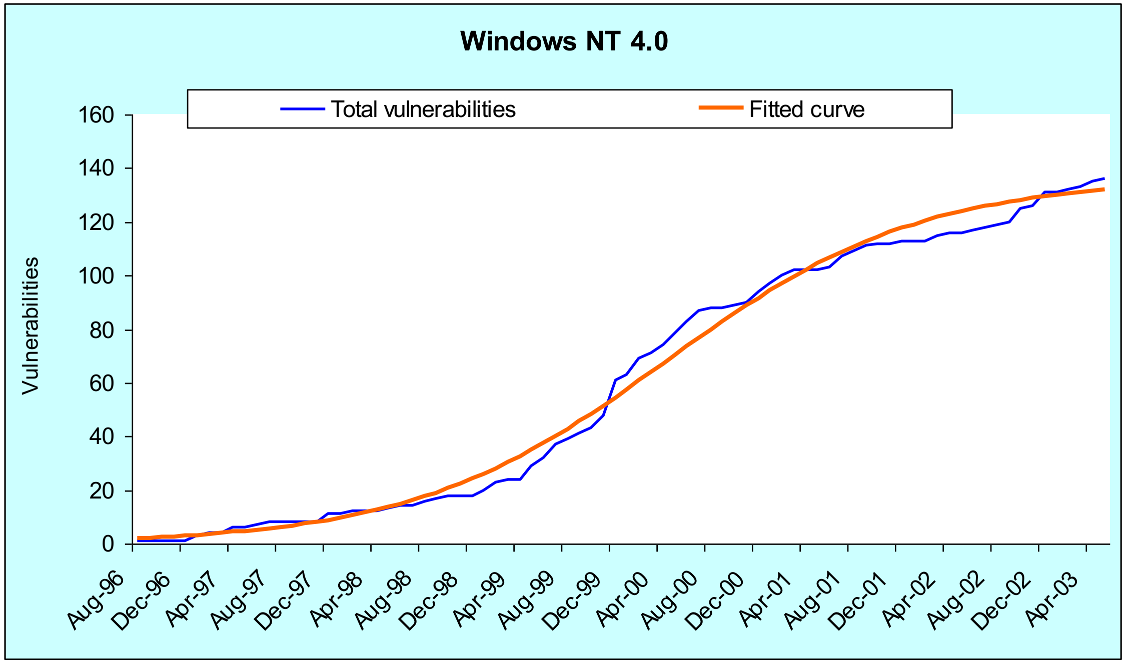
Time–based model: Windows 98



	Windows 98
A	0.004873
B	37.7328
C	0.5543
X^2	7.365
$X^2_{critial}$	60.481
P-value	$1- 7.6x10^{-11}$

Colorado State University

Time–based model: Windows NT 4.0



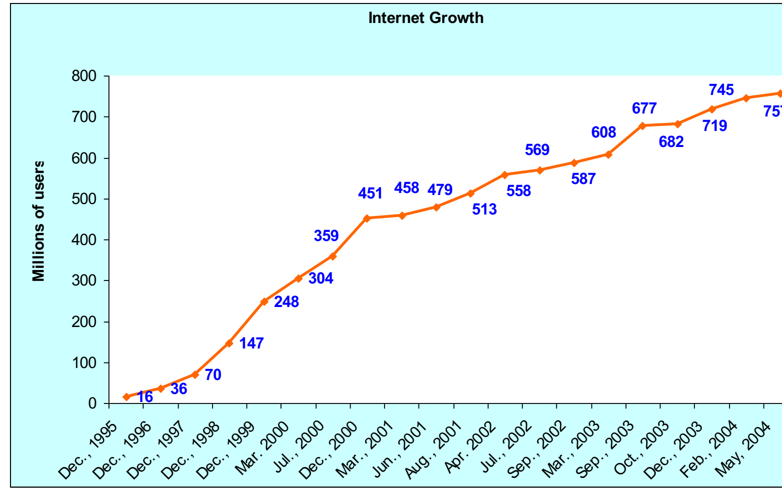
	Windows NT 4.0
A	0.000692
B	136
C	0.52288
χ^2	35.584
$\chi^2_{critial}$	103.01
P-value	0.9999973

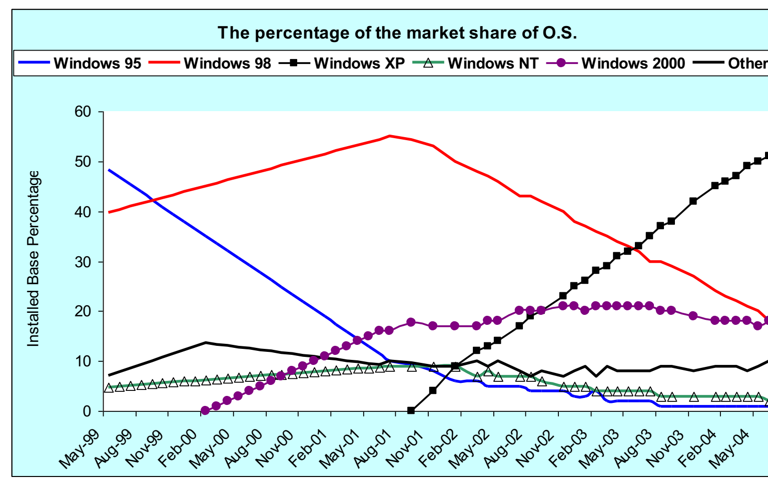
Colorado State University

Usage –vulnerability Discovery model

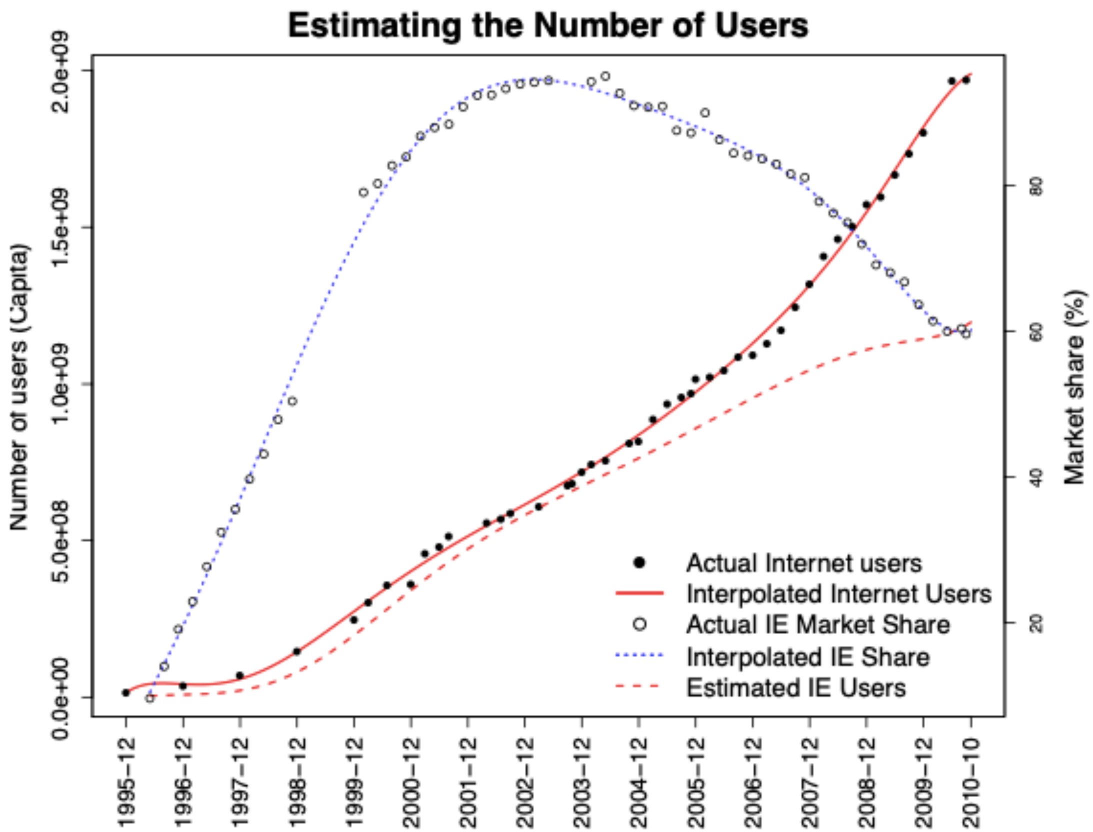
- The data:
  - The global internet population.
  - The market share of the system during a period of time.
- *Equivalent effort*
  - The real environment performs an intensive testing.
  - Malicious activities is relevant to overall activities.
  - Defined as

$$E = \sum_{i=0}^{n} (U_i \times P_i)$$





Estimating number of users



Estimating the number of IE users

QUANTITATIVE ANALYSES OF SOFTWARE VULNERABILITIES, HyunChul Joh, 2011

Software Reliability Modeling

- Applicable to general software bugs
- Key Static software metrics
  - Software size (without comments, KLOC)
  - Defect density (total defects/size)
    - Typical range Range 16 -0.1 /KLOC
    - Software evolution/reuse, requirement volatility
    - Team capabilities, extent of testing
  - Defect finding efficiency

Colorado State University

Exponential SRGM

Exponential Reliability Growth Model

- Assumption: rate of finding and removing bugs proportional to the number of bugs present at time t.

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

Which yields

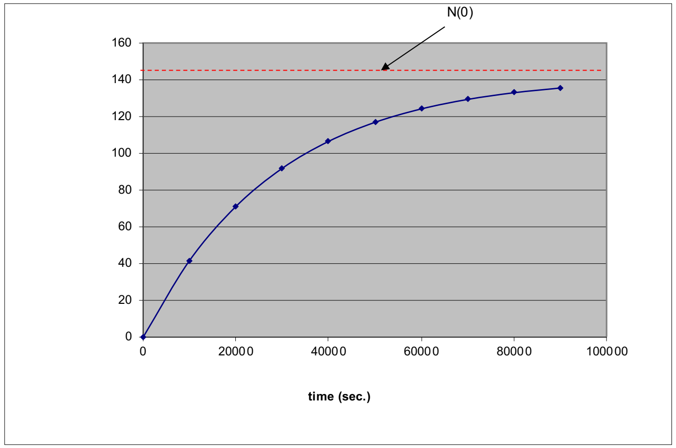
$$N(t) = N(0)e^{-\beta_1 t}$$

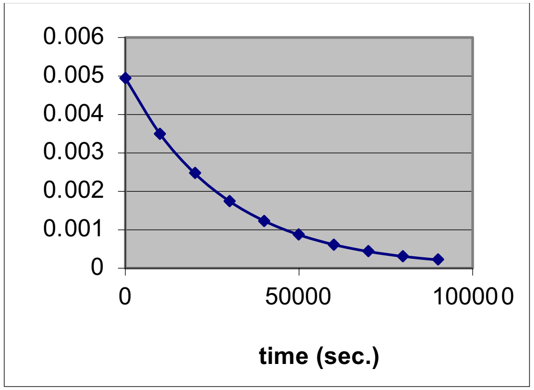
- Cumulative number of defects found is

$$N(0)(1 - e^{-\beta_1 t})$$

- Defect finding rate is

$$N(0)e^{-\beta_1 t}$$



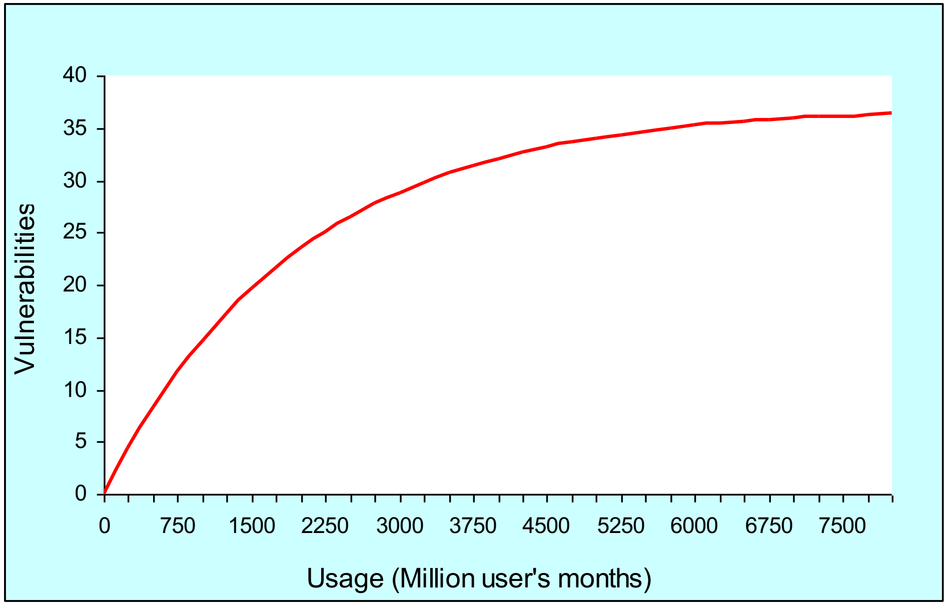


- N(0) may be estimated using defect density and size
- β_1 depends to defect finding efficiency

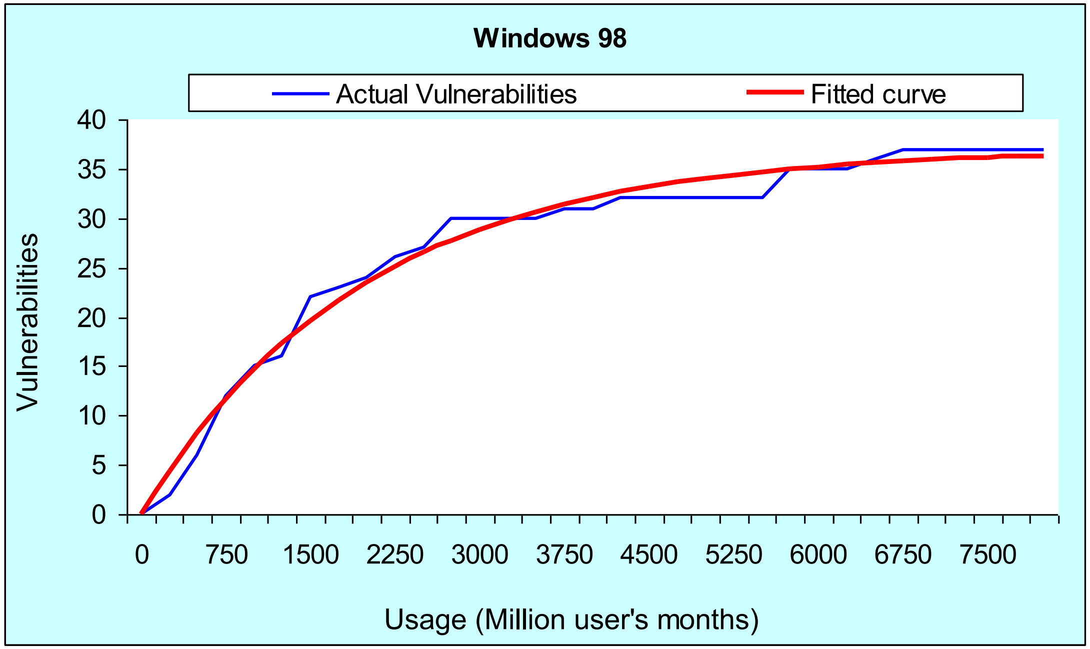
Colorado State University

Usage –vulnerability Discovery model

- The model: growth with effort.
- Growth model based on the exponential SRGM [Musa].
- Time is eliminated.
- $y = N(0)(1 - e^{-\beta_1 E})$



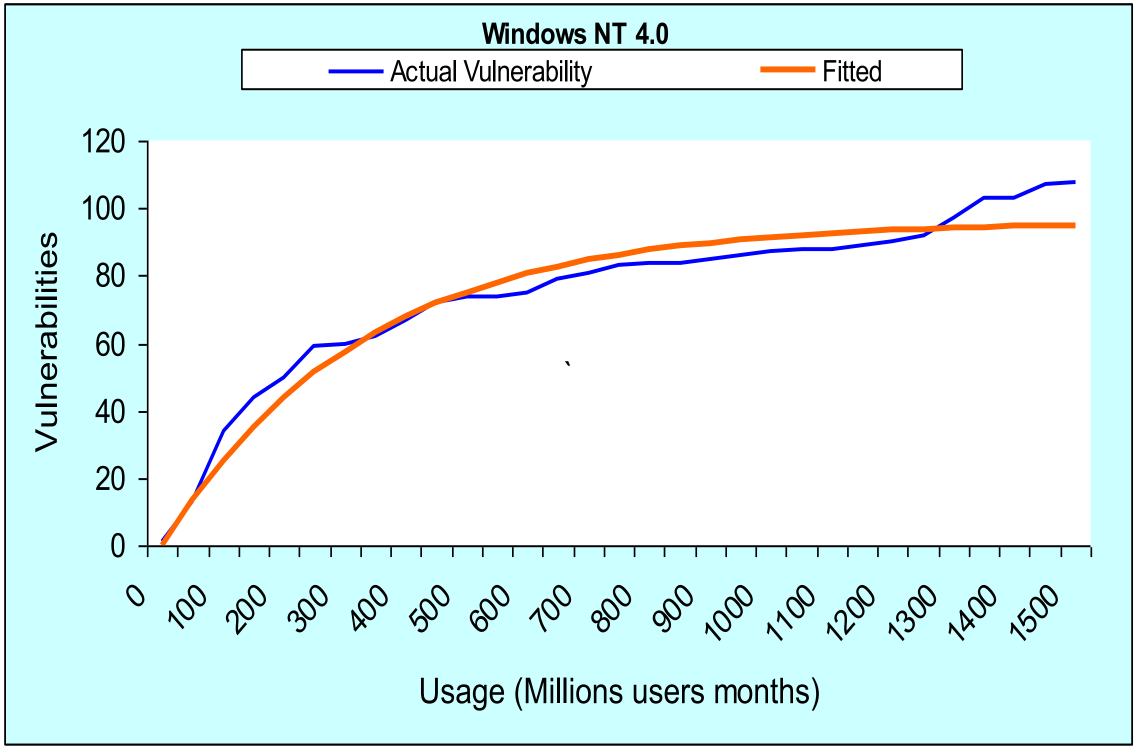
Effort-based model: Windows 98



	Windows 98
B	37
λ_{vu}	0.000505
X^2	3.510
$X^2_{critial}$	44.9853
P-value	$1- 3.3\text{x}10^{-11}$

Colorado State University

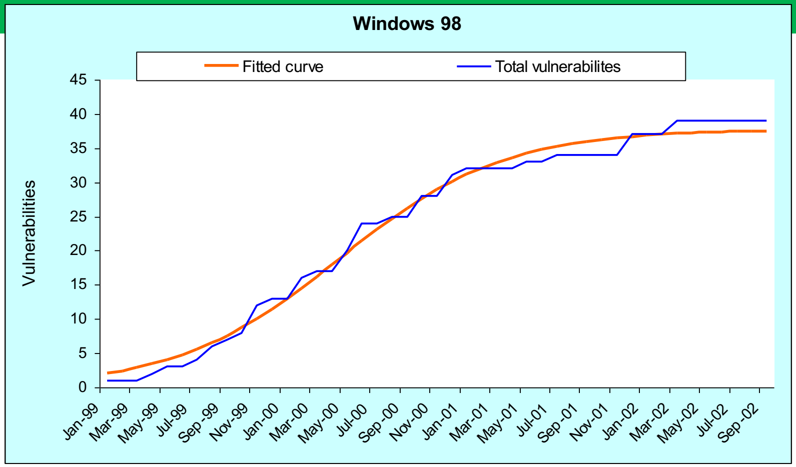
Effort-based model: Windows NT 4.0



	Win NT 4.0
B	108
λ_{vu}	0.003061
X^2	15.05
$X^2_{critial}$	42.5569
P-value	0.985

Colorado State University

Discussion



- Excellent fit for Windows 98 and NT 4.0.
- Model fits data for all OSs examined.
- Deviation from the model caused by overlap:
  - Windows 98 and Windows XP
  - Windows NT 4.0 and Windows 2000
- Vulnerabilities in shared code may be detected in the newer OS.
- Need: approach for handling such overlap

Colorado State University

Non-linear regression with Solver

- Excel has the capability to solve linear (and often nonlinear) programming problems.
- The SOLVER tool in Excel:
  - May be used to solve linear and nonlinear optimization problems
  - Allows integer or binary restrictions to be placed on decision variables
  - Can be used to solve problems with up to 200 decision variables
  - The SOLVER Add-in is a Microsoft Office Excel add-in program that is available when you install Microsoft Office or Excel.
  - To use the Solver Add-in, however, you first need to load it in Excel. The process is slightly different for Mac or PC users.

Classic Optimization Problem

- Linear Programming, Non-Linear Programming etc.
- Specified
  - Objective function: minimize or maximize
  - Constraints: equalities, inequalities
- Generally solution is iterative
- Excel Solver algorithms
    - Simplex method is used for solving linear problems
    - GRG solver for solving smooth nonlinear problems
    - Evolution solver uses genetic algorithms

Initial Values

- Start with some initial values and the gradually iterate towards optimal.
- When 3 or more parameters are used, it is best to start with some good initial guesses.
- Algorithm may get stuck at a local minimum/maximum
- Repeat with diverse initial guesses.

Example

- Example:
  - w95exmple.xlsx

$$y = \frac{B}{BCe^{-ABt} + 1}$$

- Decision variables: 3 parameter values.
- Objective Function: Sum of squares of errors between actual vs predicted values
- Constraints: all parameters must be positive

Vulnerability density and defect density

- Defect density
  - Valuable metric for planning test effort
  - Used for setting release quality target
  - Some data is available
  - Depends on various factors, may be stable for a team/process
- Vulnerabilities are a class of defects
  - Vulnerability data is in the public domain.
  - Is vulnerability density a useful measure?
  - Is it related to defect density?
    - Vulnerabilities = 5% of defects [Longstaff]?
    - Vulnerabilities = 1% of defects [Anderson]?
- Can be a major step in measuring security.

Vulnerability density and defect density

- **Vul dens**: 95/98: 0.003-0.004,   NT/2000/XP: 0.01-0.02,   Apache 0.04
- **V_{KD}/D_{KD}.** about 1% for client OSs, Higher for HTTP servers, server OSs

System	MSLOC	Known Defects (1000s)	D_{KD} (/Kloc)	Known Vulner - abilies	V_{KD} (/Kloc)	Ratio V_{KD} /D_{KD}
Win 95	15	5	0.33	46	0.0031	0.92%
NT 4.0 server	16	10	0.625	162	0.0101	1.62%
Win 98	18	10	0.556	84	0.0047	0.84%
Win2000	35	63	1.8	508	0.0145	0.81%
Win XP	40	106.5*	2.66*	728	0.0182	0.68%*
Apache HTTP 2006	227 (Unix)	4148	18.27	96	0.423	2.32%
Firefox	2.5	24,027	9.61	134	0.0536	0. 557%

MS Thesis Woo, 2006

Colorado State University

Some notes of caution

- We can never really know the actual number of
  - ordinary software defects
  - Vulnerabilities

- We can only count the bugs/vulnerabilities that are known.
- Some methods exist to estimate the number of defects not yet found:
  - SRGMs
  - Defect found-coverage relationship (Malaiya et al 94, 98)
- Similar methods may be devised for vulnerabilities

Factors Impacting Vulnerabilities

Number of vulnerabilities:

- Software Code Size: assuming vulnerability density remains about the same
- Fraction of code that handles access in/out: higher densities for web servers, browsers
- Software age/reuse: vulnerabilities are discovered and removed with time, new code injects new vulnerabilities

Vulnerability discovery rate:

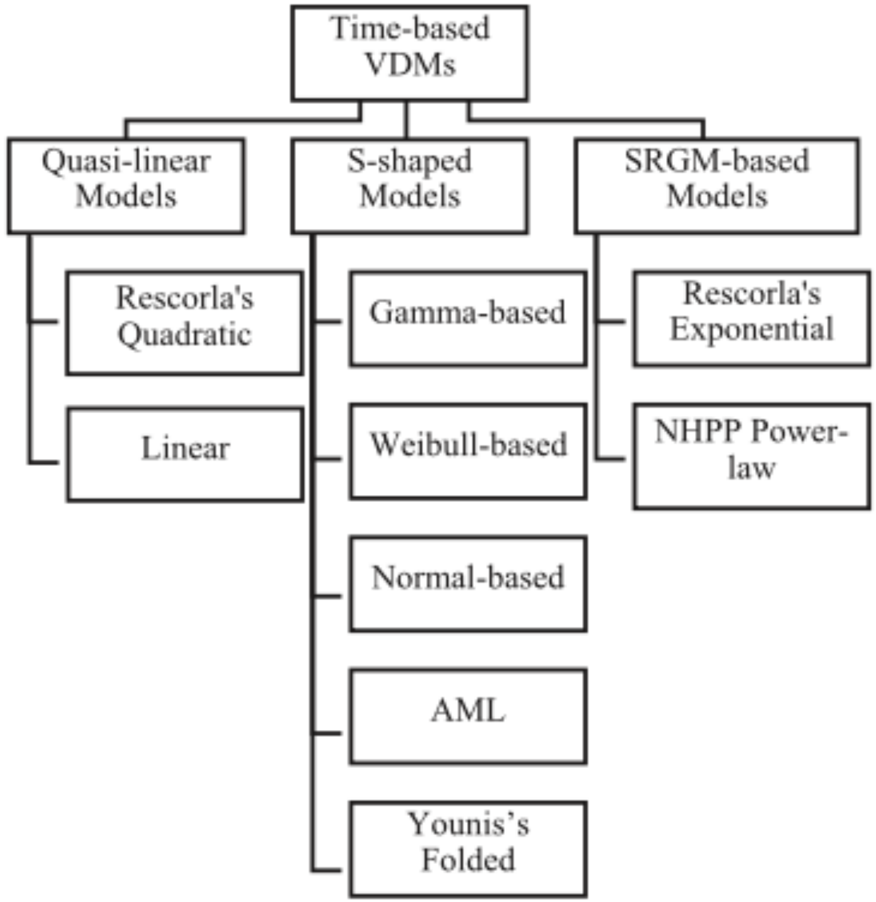
- Installed system base: higher base makes the product more attractive
- Vulnerability discovery tools/expertise

Summary and conclusions

We have introduced:

- Models:
  - Time – vulnerability model.
  - Usage – vulnerability model.
  - Both models shown acceptable goodness of fit.
    - Chi-square test.
- Measurements:
  - vulnerability density.
  - Vulnerability density vs. defect density.

Time-based VDMs



- Classification of Time-based VDMs.

Vulnerability Discovery Models

Model	Equation
NHPP Power-law (Movahedi et al., 2018)	$\Omega(t) = (\beta^{-\alpha}).t^{\alpha}$
Gamma-based VDM (Joh and Malaiya, 2014)	$\Omega(t_0) = \int_{t=0}^{t_0} \frac{\gamma}{\Gamma(\alpha)\beta^{\alpha}} t^{\alpha-1} e^{-\frac{t}{\beta}} dt$
Weibull-based VDM (Kim et al., 2007)	$\Omega(t) = \gamma\{1 - e^{-(\frac{t}{\beta})^{\alpha}}\}$
AML VDM (Alhazmi and Malaiya, 2005)	$\Omega(t) = \frac{B}{BCe^{-ABt}+1}$
Normal-based VDM (Joh and Malaiya, 2014)	$\Omega(t) = \frac{\gamma}{1+e^{-\frac{(t-\mu)}{\sigma}}}$
Rescorla Exponential (RE) (Rescorla, Jan. 2005)	$\Omega(t) = \gamma(1 - e^{-\lambda t})$
Rescorla Quadratic (RQ) (Rescorla, Jan. 2005)	$\Omega(t) = \frac{At^2}{2} + Bt$
Younis Folded (YF) (Younis et al., 2011)	$\Omega(t) = \frac{\gamma}{2}\{\mathrm{erf}(\frac{t-\tau}{\sqrt{2}\sigma}) + \mathrm{erf}(\frac{t+\tau}{\sqrt{2}\sigma})\}$
Linear Model (LM) (Alhazmi and Malaiya, 2006)	$\Omega(t) = At + B$

Table of models and their equations

Yazdan Movahedi, Michel Cukier, Ilir Gashi, Vulnerability prediction capability: A comparison between vulnerability discovery models and neural network models, Computers & Security,, Volume 87, 2019.

Seasonality in Vulnerability Discovery

Colorado State University

Seasonality in Vulnerability Discovery

- Vulnerability Discovery Model (VDM):
  - a probabilistic methods for modeling the discovery of software vulnerabilities [Ozment]
  - Spans a few years: introduction to replacement
- Seasonality: periodic variation
  - well known statistical approach
  - quite common in economic time series
    - Biological systems, stock markets etc.

Halloween indicator:
Low returns in May-Oct.

Examining Seasonality

- Is the seasonal pattern statistically significant?
- Periodicity of the pattern
- Analysis:
  - Seasonal index analysis with test
  - Autocorrelation Function analysis

χ^2

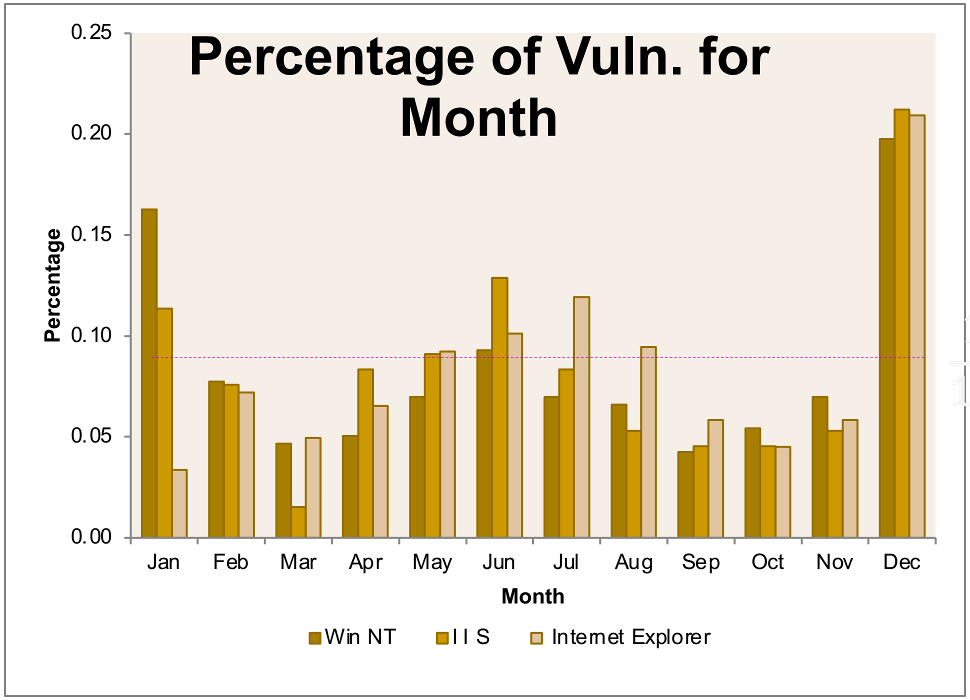
- Significance
  - Enhance VDMs' predicting ability
- Annual and Weekly seasonality

35

Annual: Prevalence in Month

Vulnerabilities Disclosed

	WinNT '95~'07	IIS '96~'07	IE '97~'07
Jan	42	15	15
Feb	20	10	32
Mar	12	2	22
Apr	13	11	29
May	18	12	41
Jun	24	17	45
Jul	18	11	53
Aug	17	7	42
Sep	11	6	26
Oct	14	6	20
Nov	18	7	26
Dec	51	28	93
Total	258	132	444
Mean	21.5	11	37
s.d.	12.37	6.78	20.94



36

Colorado State University

Seasonal Index

Seasonal Index Values

	WinNT	IIS	IE
Jan	1.95	1.36	0.41
Feb	0.93	0.91	0.86
Mar	0.56	0.81	0.59
Apr	0.60	1.00	0.78
May	0.84	1.09	1.11
Jun	1.12	1.55	1.22
Jul	0.84	1.00	1.43
Aug	0.79	0.64	1.14
Sep	0.51	0.55	0.70
Oct	0.65	0.55	0.54
Nov	0.84	0.64	0.70
Dec	2.37	2.55	2.51
χ_c^2	19.68	19.68	19.68
χ_s^2	78.37	46	130.43
p-value	3.04e-12	3.23e-6	1.42e-6

- Seasonal index: measures how much the average for a particular period tends to be above (or below) the expected value
- H_0: no seasonality is present. We will evaluate it using the monthly seasonal index values given by [4]:

$$s_i = \frac{d_i}{d}$$

where, s_i is the seasonal index for i^{th} month, d_i is the mean value of i^{th} month, d is a grand average

[4] Hossein Arsham. Time-Critical Decision Making for Business Administration. Available: http://home.ubalt. edu/ntsbarsh/Business-stat/stat-data/Forecast.htm#rseasonindx

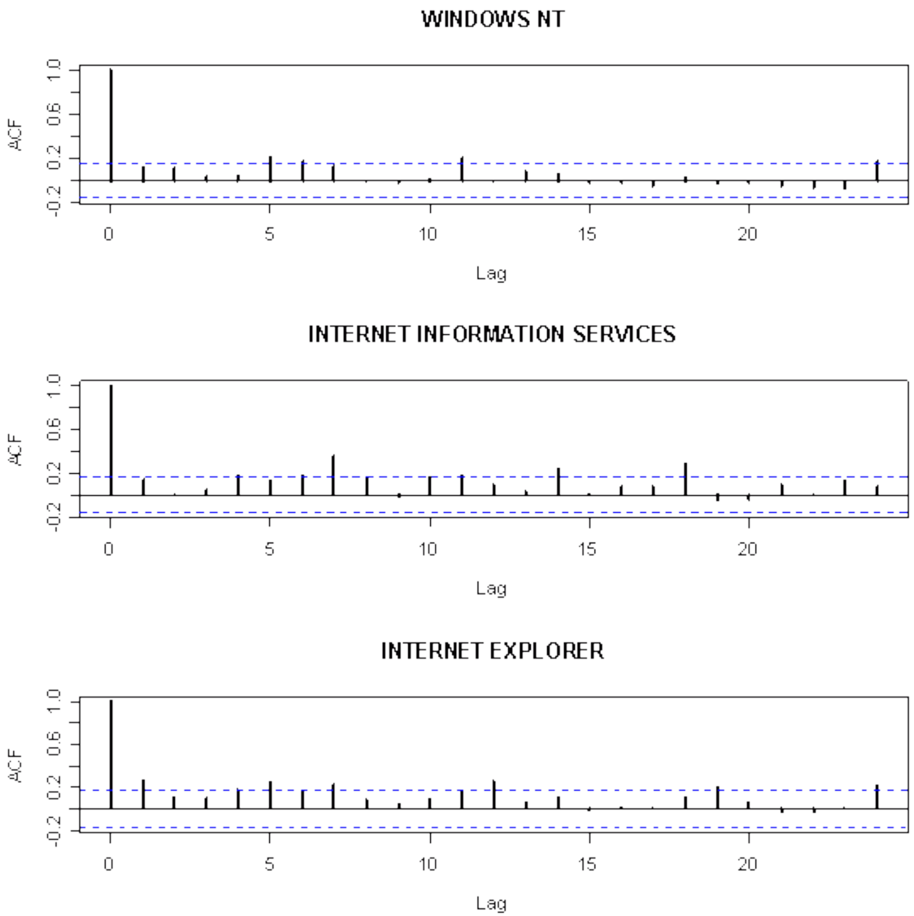
Autocorrelation function (ACF)

- Plot of autocorrelations function values
- With time series values of $z_b, z_{b+1}, \ldots, z_n$, the ACF at lag k, denoted by r_k, is [5]:
- $$r_k = \frac{\sum_{t=b}^{n-k}(z_t - \bar{z})(z_{t+k} - \bar{z})}{\sum_{t=b}^{n}(z_t - \bar{z})^2}$$
- , where $\bar{z} = \frac{\sum_{t=b}^{n} z_t}{(n - b + 1)}$
- Measures the linear relationship between time series observations separated by a lag of time units
- Hence, when an ACF value is located outside of confidence intervals at a lag t, it can be thought that every lag t, there is a relationships along with the time line

[5] B. L. Bowerman and R. T. O'connell, Time Series Forecsting: Unified concepts and computer implementation. 2nd Ed., Boston: Duxbury Press, 1987

38

Autocorrelation (ACF):Results



- Expected lags corresponding to 6 months or its multiple would have their ACF values outside confidence interval
- Upper/lower dotted lines: 95% confidence intervals.
- An event occurring at time t + k (k > 0) lags behind an event occurring at time t.
- Lags are in month.

Colorado State University

Why seasonality?

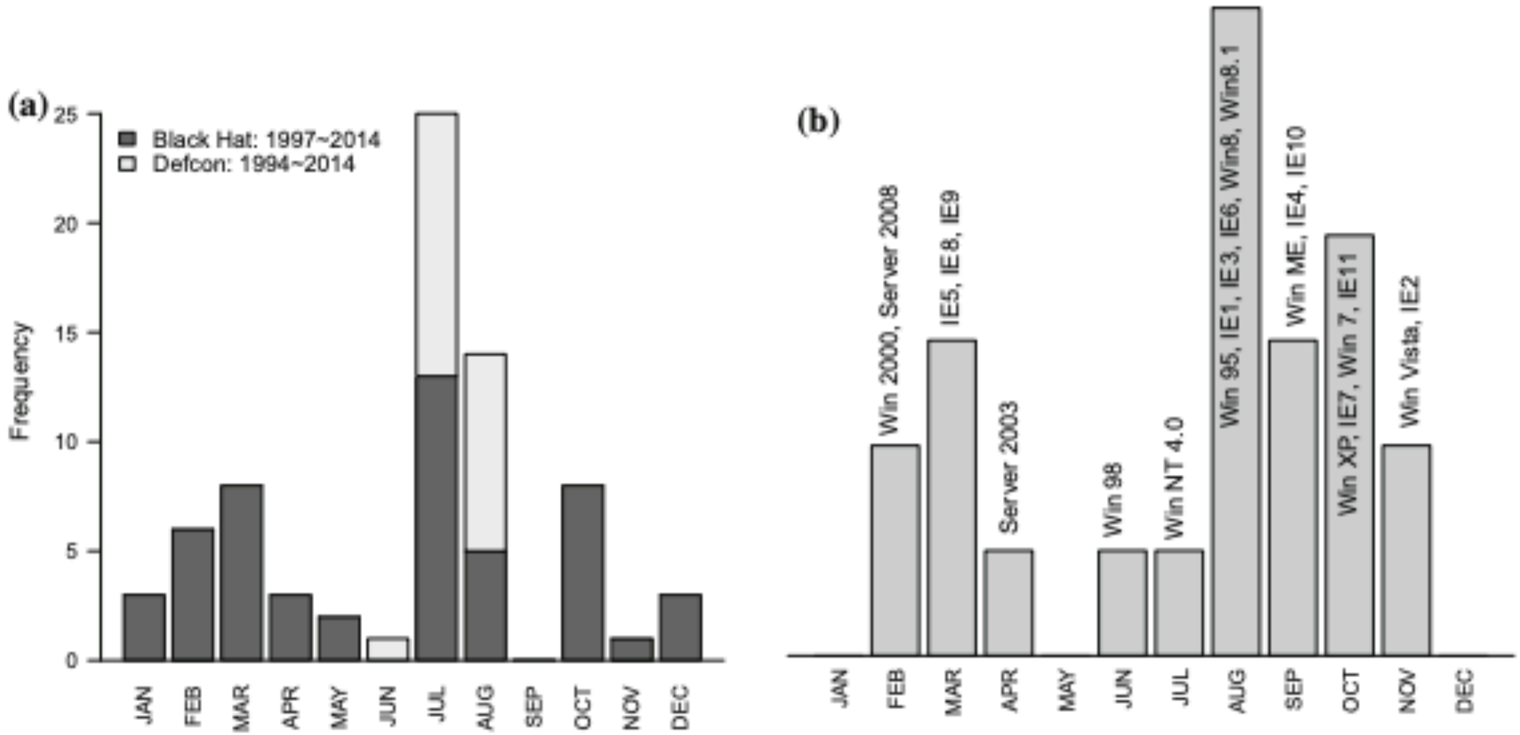


Fig. 6 Frequency of Black Hat and Defcon by month, and major Microsoft software system release time by month. **a** Black Hat and Defcon by month. **b** MS release by month

H. Joh and Y.K. Malaiya, "Periodicity in Software Vulnerability Discovery, Patching and Exploitation",International Journal of Information Security, July 2016, pp 1-18.

Colorado State University

Weekly Seasonality

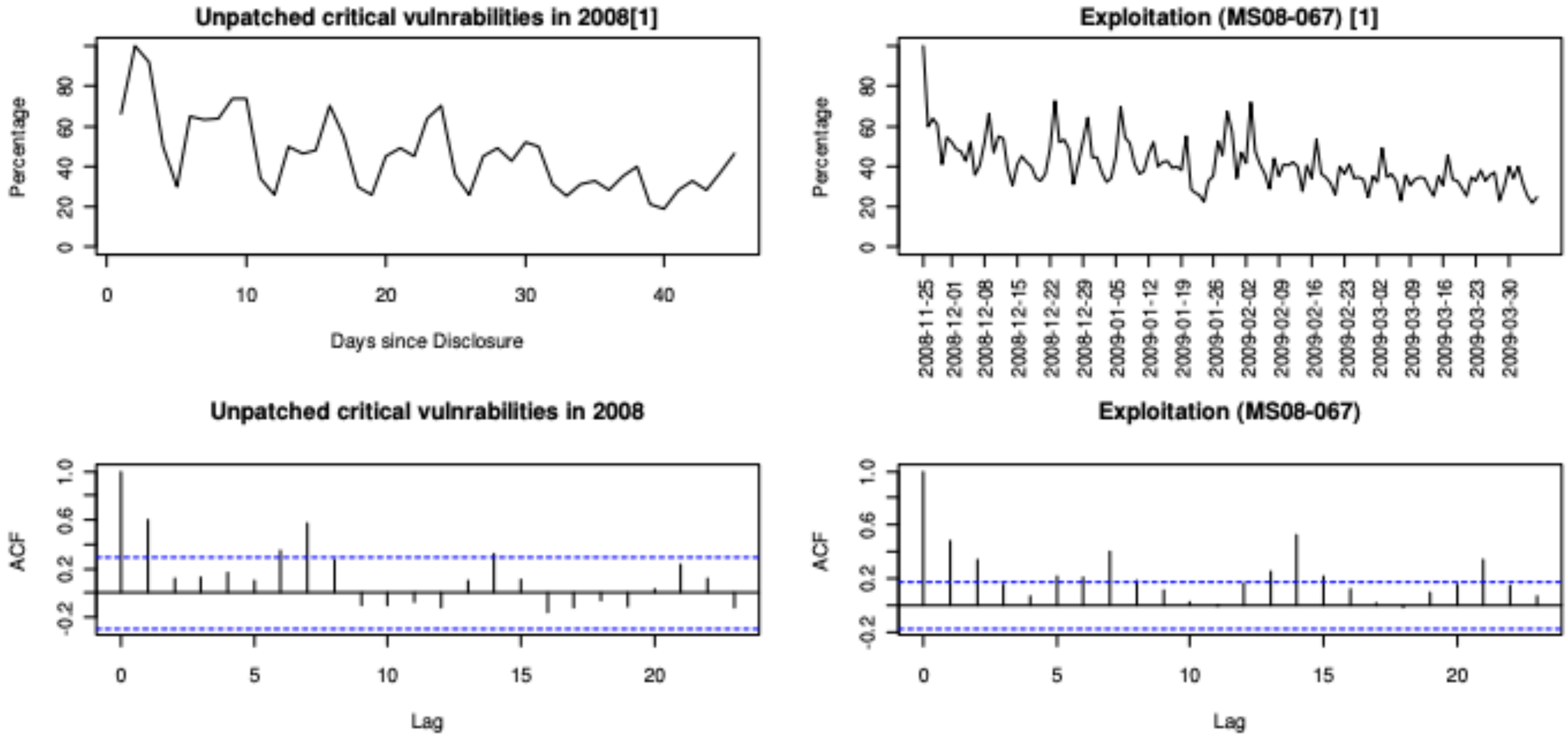


Figure 1. Run charts for unpatched critical vulnrabilities in 2008 and Exploitation with their corresponding ACFs. The upper two plots are normalized using the maximum value as 100%. In the bottom two plots, legs are in day.

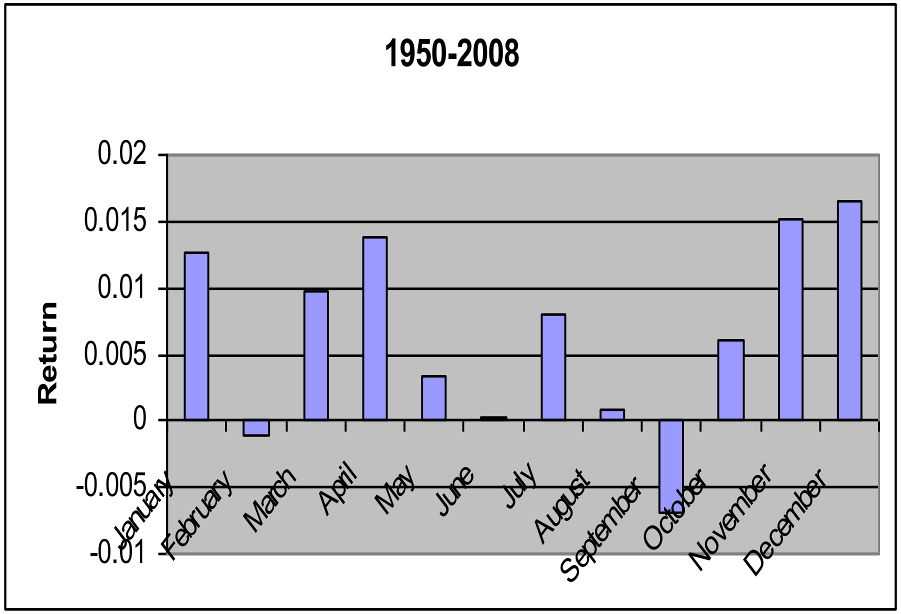
H. Joh, S. Chaichana and Y. K. Malaiya, "[Short-term Periodicity in Security Vulnerability Activity]" Proc. Int. Symp. Software Reliability Eng. (ISSRE), FA, November 2010, pp. 408-409

Colorado State University

41

Halloween Indicator

- "Also known as "Sell in May and go away"
- Global (1973-1996):
  - Nov.-April: 12.47% ann., st dev 12.58%
  - 12-months:10.92%, st. dev. 17.76%
- 36 of 37 developing/developed nations
- Data going back to 1694
- "No convincing explanation"



Jacobsen, Ben and Bouman, Sven,The Halloween Indicator, 'Sell in May and Go Away': Another Puzzle(July 2001). Available at SSRN: http://ssrn.com/abstract=76248

Quantitative Security

Colorado State University
Yashwant K Malaiya

CS 559

Multi-version Systems

CSU Cybersecurity Center
Computer Science Dept

- Motivation
- Software Evolution
- Multi-version Software Discovery Model
  - Apache, Mysql and Win XP data

44

Motivation for Multi-version VDMs

- Superposition effect on vulnerability discovery process due to shared code in successive versions.
- Examination of software evolution: impact on vulnerability introduction and discovery
- Other factors impacting vulnerability discovery process not considered before

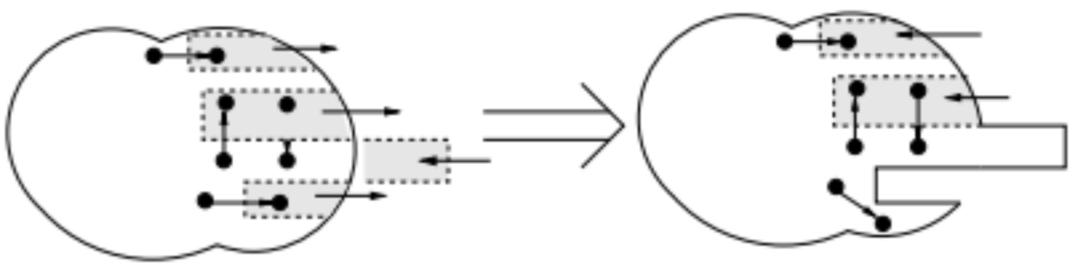
45

Software Reuse

- New software projects use both new and reused blocks.
  - New blocks have a higher defect density because they have undergone less testing.
  - Reused blocks are more reliable.
  - Some defects may be introduced at the new/reused block interface.
  - Overall defect density is weighted average of the two.
  - Encounter rate during execution depends on weighted usage

Software Evolution

- The modification of software during maintenance or development:
  - fixes and feature additions.
  - Influenced by competition
- Code decay and code addition introduce new vulnerabilities
- Successive version of a software can share a significant fraction of code.

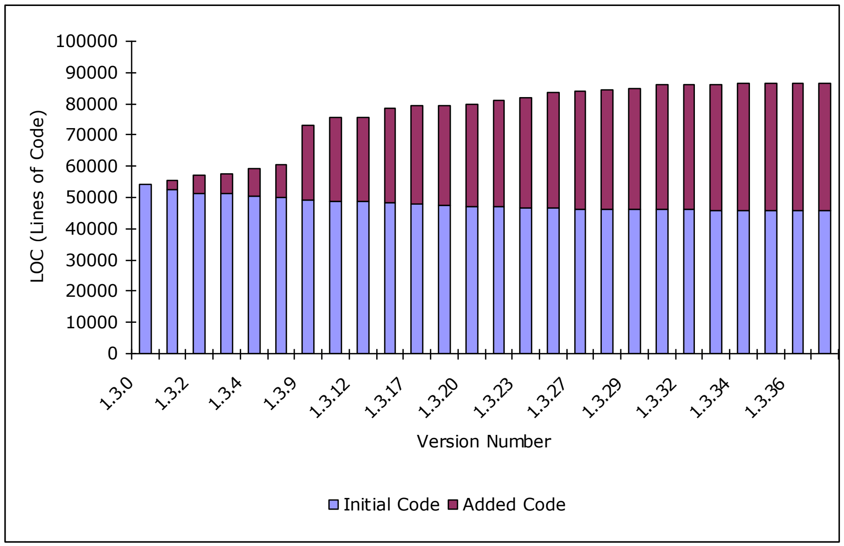


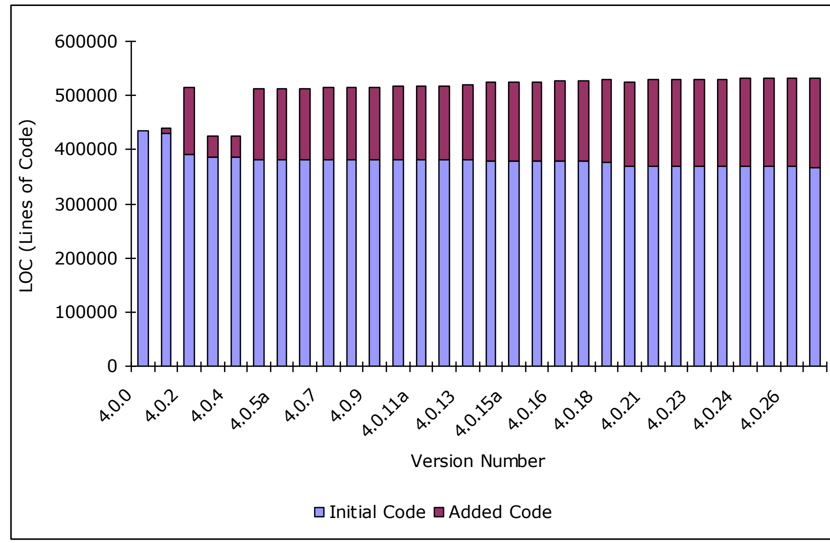
Y. K. Malaiya and J. Denton "Requirement Volatility and Defect Density,"
Proc. IEEE Int. Symp. Software Reliability Engineering, Nov. 1999, pp. 285-294.

Colorado State University

47

Software Evolution: Apache & Mysql

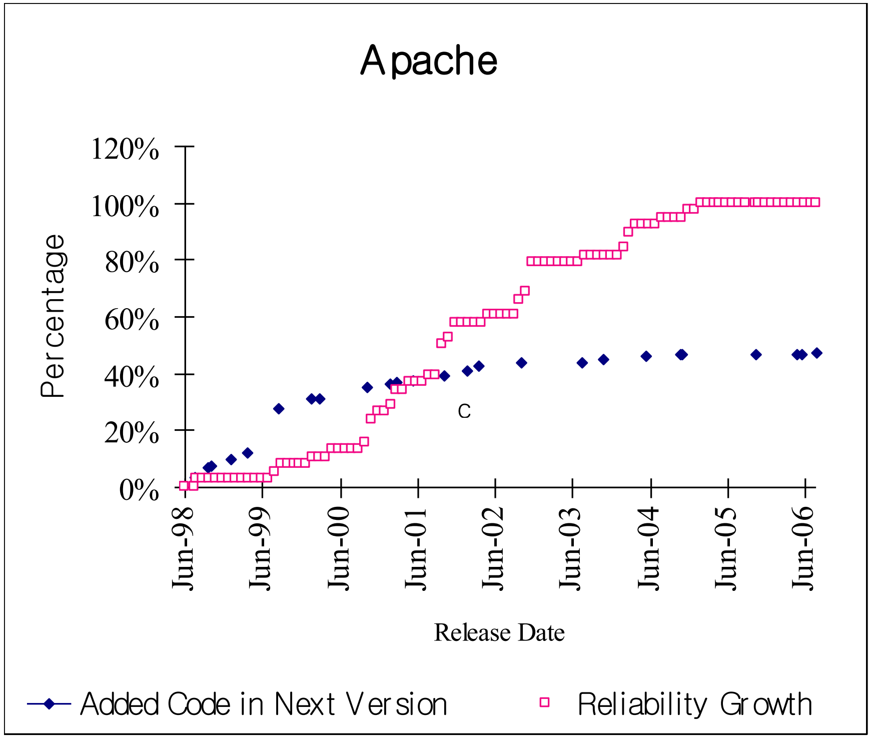


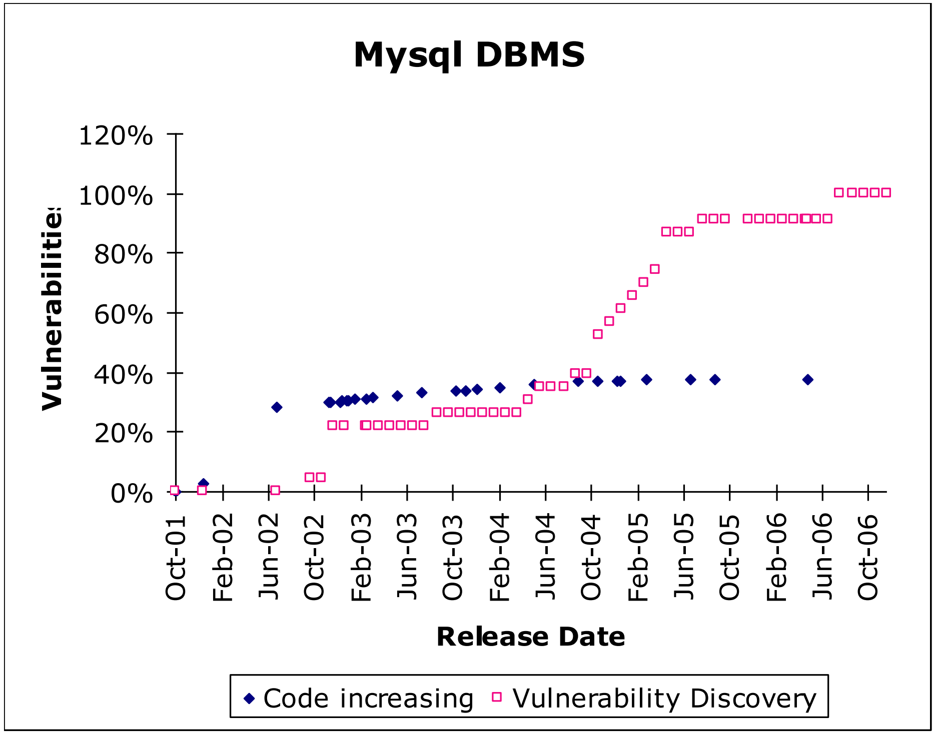


Modification: Apache 43%, Mysql 31%

J. Kim, Y. K. Malaiya and I. Ray, "Vulnerability Discovery in Multi-Version Software Systems," Proc. 10th IEEE Int. Symp. on High Assurance System Engineering (HASE), Dallas, Nov. 2007, pp. 141-148

Vulnerability Discovery & Evolution:

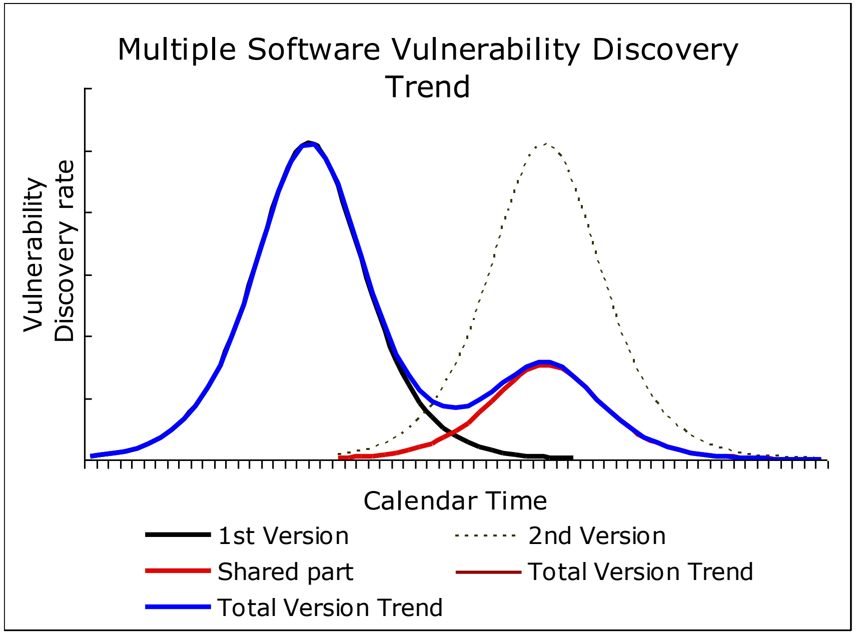




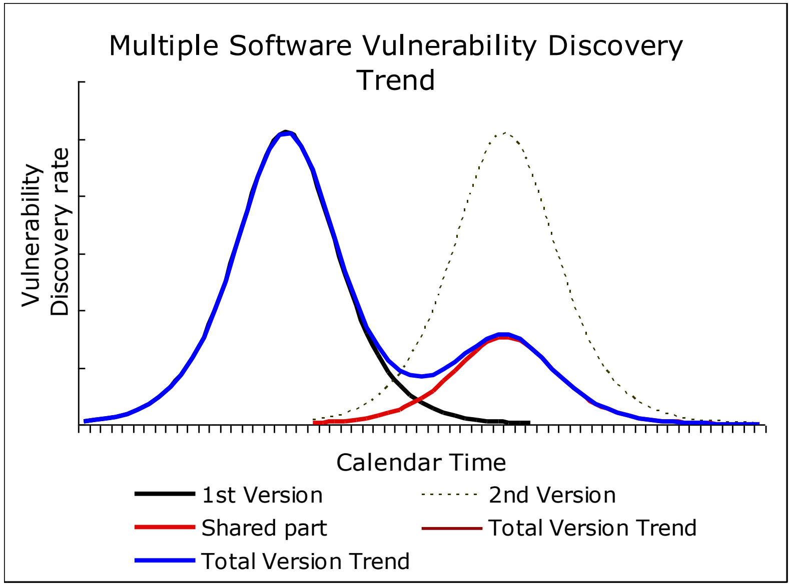
Some vulnerabilities are in added code, many are inherited from precious versions.

Code Sharing & Vulnerabilities

- Observation
  - Vulnerability increases after saturation in AML modeling
- Accounting for Superposition Effect
  - Shared components between several versions of software



Multi-version Vulnerability Discovery

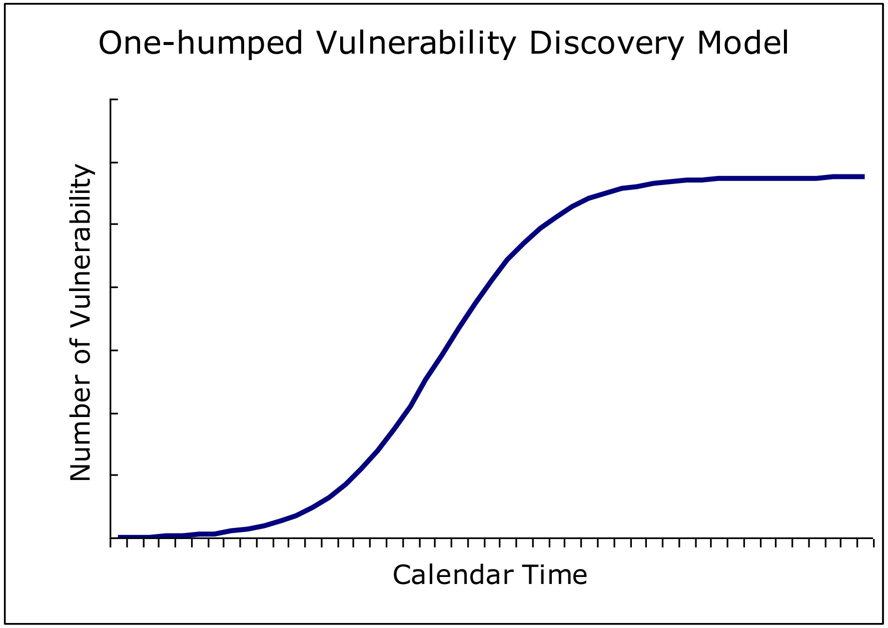


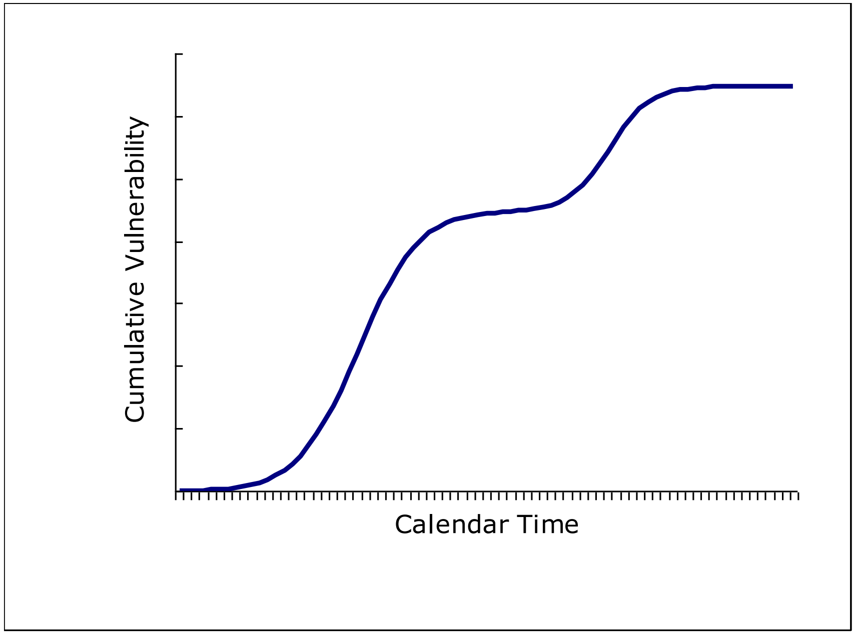
$$\Omega(t) = \frac{B}{BCe^{-ABt}+1} + \alpha\frac{B'}{B'C'e^{-A'B'(t-s)}+1}$$

	Previous Version	Next Version	Shared Code Ratio α
Apache	1.3.24 (3-21-2002)	2.0.35 (4-6-2002)	20.16%
Mysql	4.1.1 (12-1-2003)	5.0.0 (12-22-2003)	83.52%

51

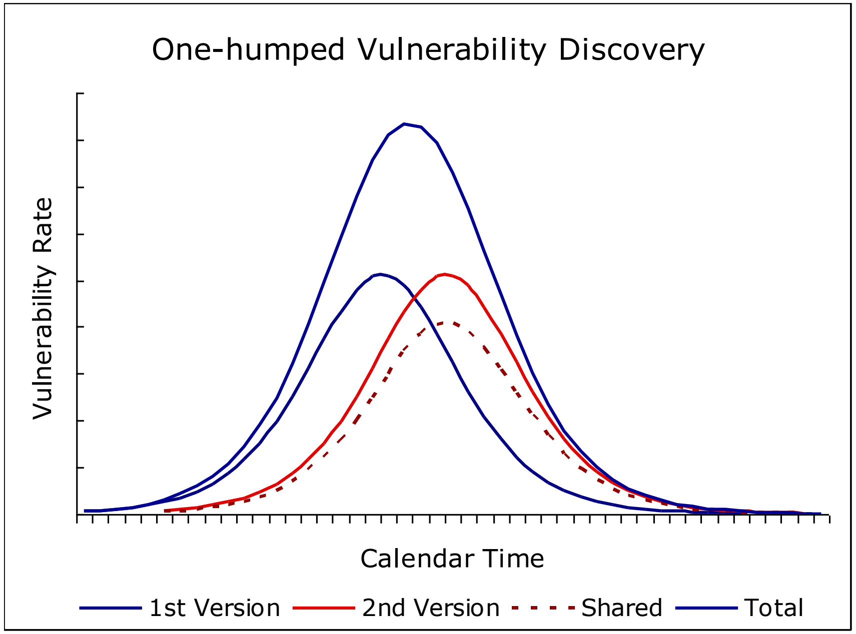
One vs Two Humps

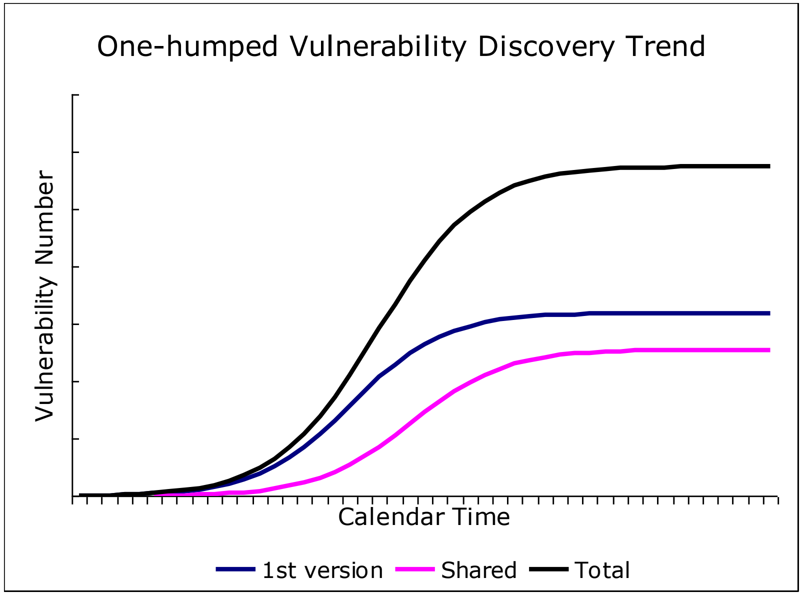




Superposition affect

Multi-version Vulnerability Discovery





- May result in a single hump with prolonged linear period

Evolving Programs

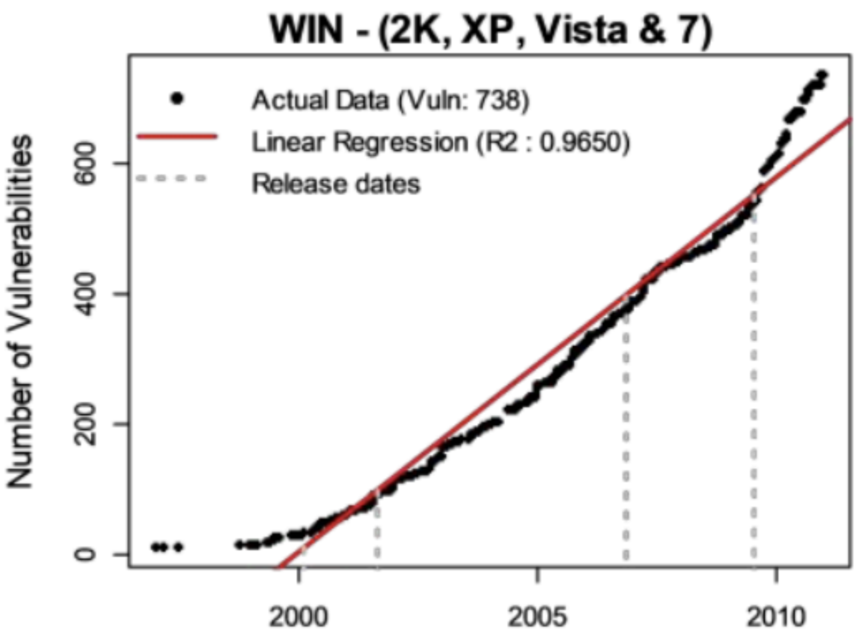
Gradually evolving software

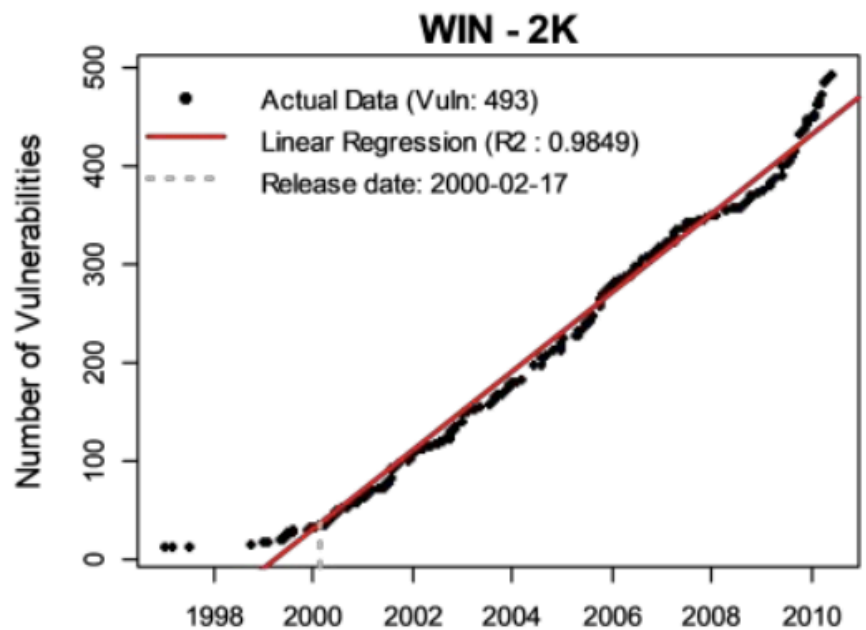
Software evolves in each version.

- Existing code fixed
  - some vulnerabilities found and patched
- Code added for increasing functionality
  - New vulnerabilities injected
  - Total number of vulnerabilities may remain about the same
- Overall code size keeps increasing
  - Vulnerability discovery rate may remain stable

Linear model

- Because of nearly continuous evolution, the linear phase may get stretched.

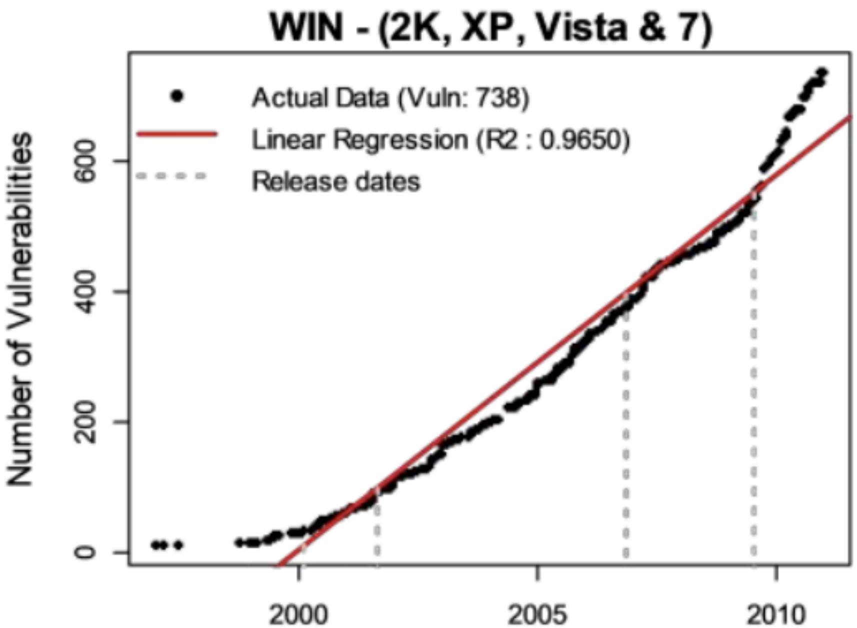


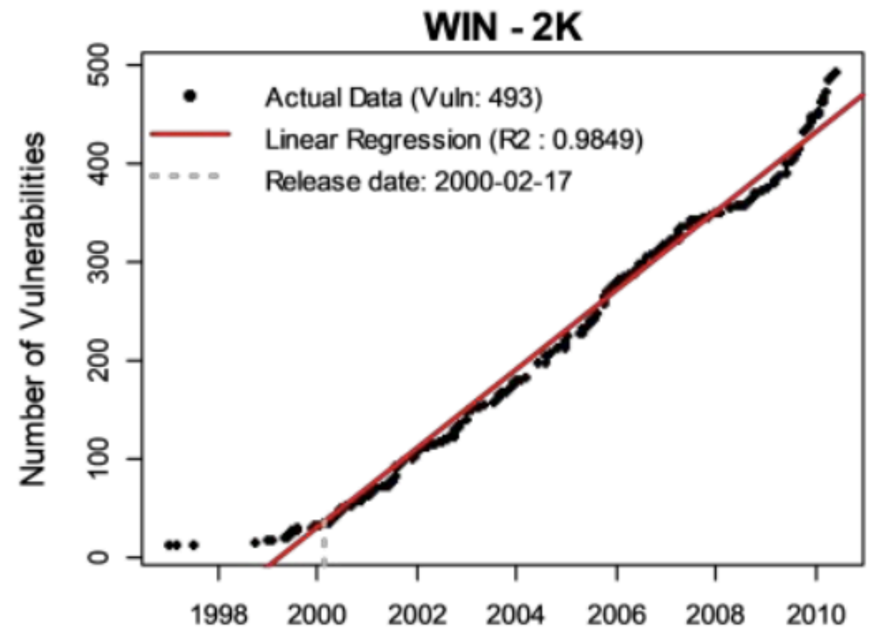


- If the evolution rate is steady, the size of the pool of undiscovered vulnerabilities stays the same.
- If the market share is steady, the number of vulnerability finders remains steady.

Joh’s thesis

Linear model

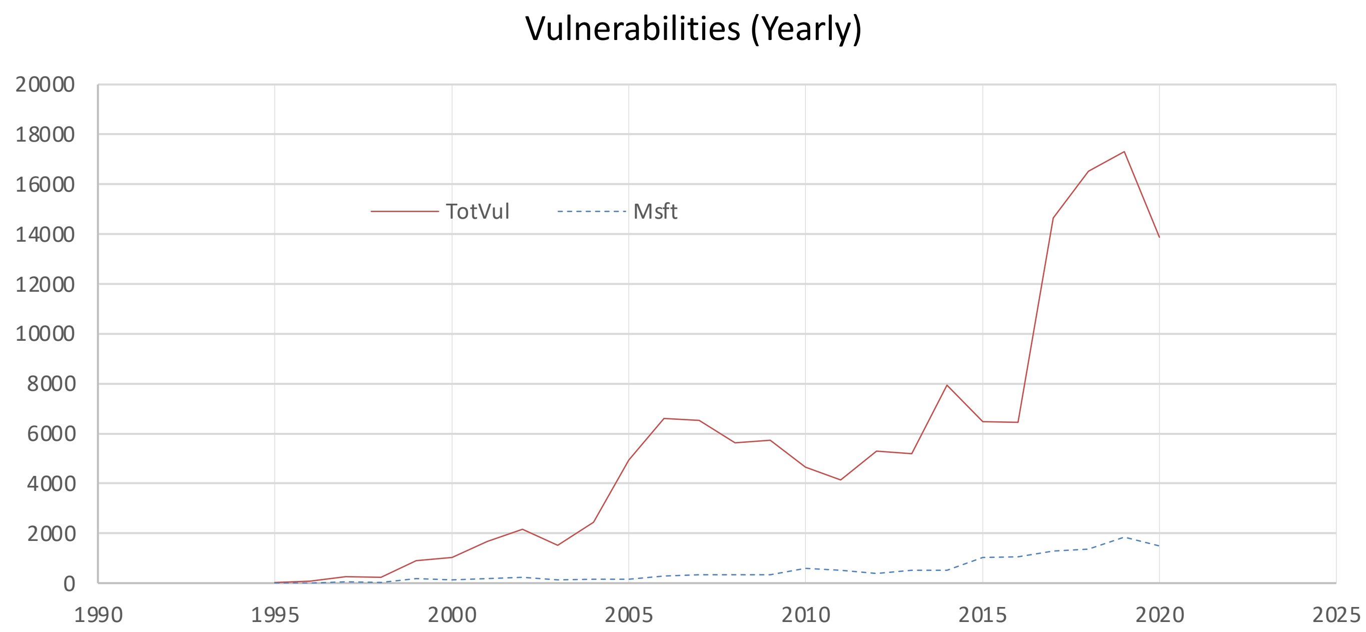




- Four Windows releases: 500 vulnerabilities during July 1998-July 2009
- Size: 35-50 M LOC
- Slope = about 45 vulnerabilities/year
- Further investigation is needed.

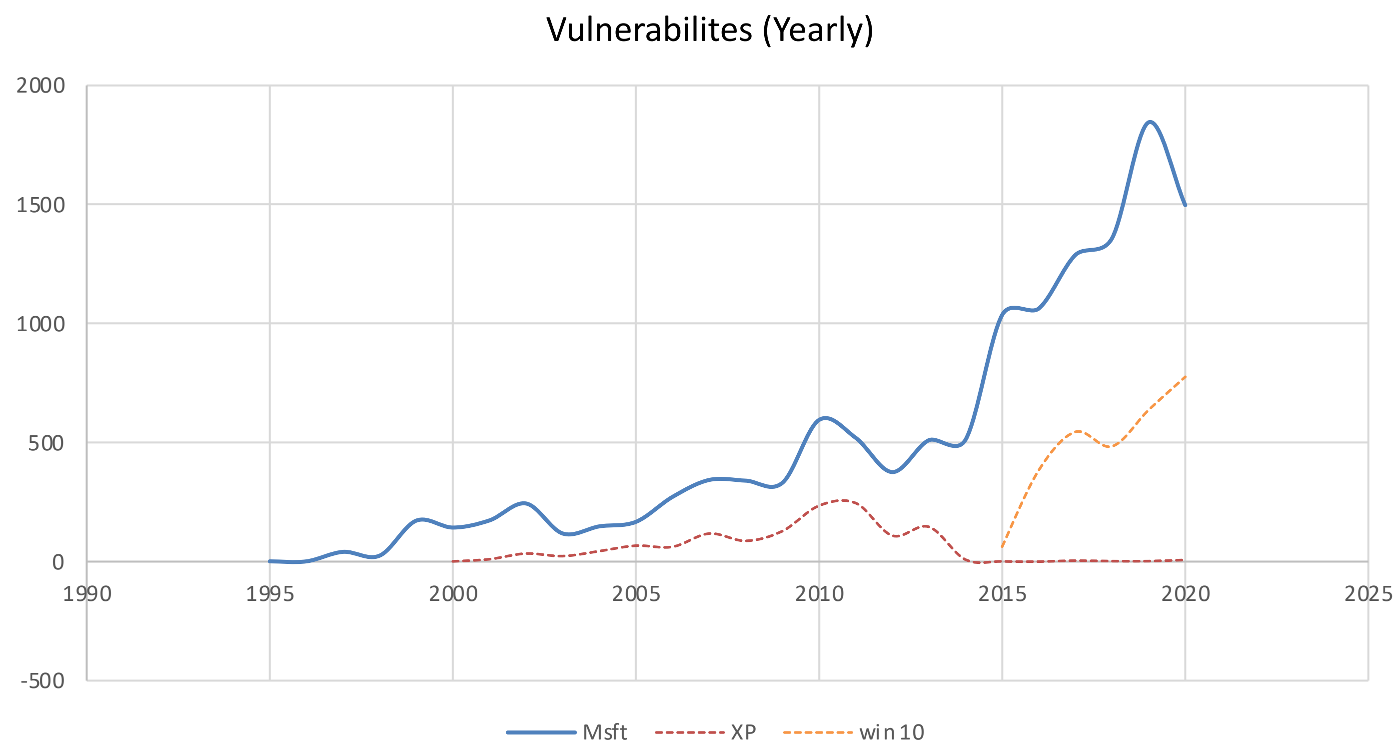
Data from Joh's thesis

Long Term Trends



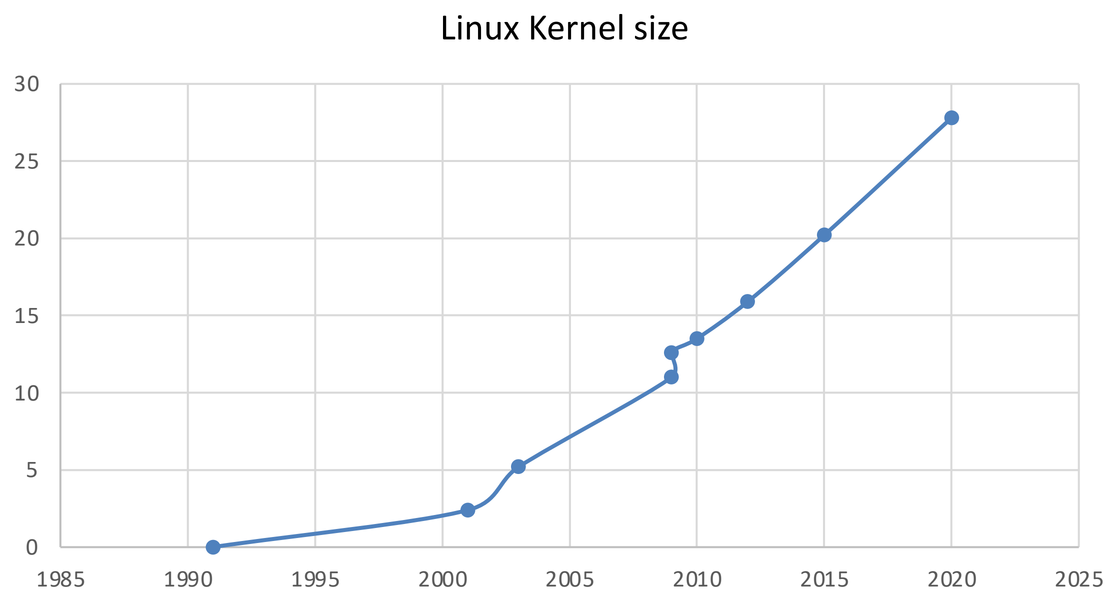
- Long term Trends: Total vulnerabilities, Microsoft products

Long Term Trends



- Long term Trends: Microsoft products, Win XP, Win 10

Long Term Trends



- Size evolution: Linus kernel

Long term trends

Likely factors that affect long-term trends

- Better understanding of safer coding practices
  - Fewer vulnerabilities injected?
- Better vulnerability discovery tools (fuzzers) and more finders
  - Higher vulnerability discovery rates

Vulnerability Discovery and Risks

What factors impact risk?

- Not the vulnerabilities that have been found and patched
- Vulnerabilities that have been discovered but not patched
  - Before disclosure: black hat people/organizations
  - after disclosure: when patch development is taking time
- Vulnerabilities with patches, but patches not applied
- Statistical modeling may be needed for assessing probability of breaches