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
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
• Each vulnerability is recorded.
  – Available [NVD, vendor 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.
3 phase model S-shaped model.

- Phase 1:
  - Installed base – low.

- Phase 2:
  - Installed base – higher and growing/stable.

- Phase 3:
  - Installed base – dropping.

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

\[
y = \frac{B}{BCe^{-ABt} + 1}
\]
Alhazmi Malaiya Logistic model (AML)

\[
\frac{d\Omega}{dt} = A\Omega(B - \Omega), \quad (3)
\]

where \( \Omega \) 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

\[
\Omega(t) = \frac{B}{BCe^{-ABt} + 1}, \quad (4)
\]

Time–based model: Windows 98

Windows 98

<table>
<thead>
<tr>
<th></th>
<th>Windows 98</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.004873</td>
</tr>
<tr>
<td>B</td>
<td>37.7328</td>
</tr>
<tr>
<td>C</td>
<td>0.5543</td>
</tr>
<tr>
<td>$X^2$</td>
<td>7.365</td>
</tr>
<tr>
<td>$X^2_{critical}$</td>
<td>60.481</td>
</tr>
<tr>
<td>P-value</td>
<td>1 - 7.6x10^{-11}</td>
</tr>
</tbody>
</table>
Time–based model: Windows NT 4.0

Windows NT 4.0

Total vulnerabilities | Fitted curve

<table>
<thead>
<tr>
<th>Windows NT 4.0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.000692</td>
</tr>
<tr>
<td>B</td>
<td>136</td>
</tr>
<tr>
<td>C</td>
<td>0.52288</td>
</tr>
<tr>
<td>$\chi^2$</td>
<td>35.584</td>
</tr>
<tr>
<td>$\chi^2_{critical}$</td>
<td>103.01</td>
</tr>
<tr>
<td>P-value</td>
<td>0.9999973</td>
</tr>
</tbody>
</table>
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

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
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
- \(\beta_1\) depends on defect finding efficiency
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

<table>
<thead>
<tr>
<th></th>
<th>Windows 98</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B )</td>
<td>37</td>
</tr>
<tr>
<td>( \lambda_{vu} )</td>
<td>0.000505</td>
</tr>
<tr>
<td>( X^2 )</td>
<td>3.510</td>
</tr>
<tr>
<td>( X^2_{critical} )</td>
<td>44.9853</td>
</tr>
<tr>
<td>( P)-value</td>
<td>1 - 3.3 \times 10^{-11}</td>
</tr>
</tbody>
</table>

Usage (Million user's months)
Effort-based model: Windows NT 4.0

<table>
<thead>
<tr>
<th></th>
<th>Win NT 4.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B$</td>
<td>108</td>
</tr>
<tr>
<td>$\lambda_{vu}$</td>
<td>0.003061</td>
</tr>
<tr>
<td>$\chi^2$</td>
<td>15.05</td>
</tr>
<tr>
<td>$\chi^2_{critical}$</td>
<td>42.5569</td>
</tr>
<tr>
<td>P-value</td>
<td>0.985</td>
</tr>
</tbody>
</table>

- **Usage (Millions users months)**
- **Vulnerabilities**

![Graph showing the actual and fitted vulnerabilities for Windows NT 4.0](image)

**Windows NT 4.0**

- Actual Vulnerability
- Fitted
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
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

\[
y = \frac{B}{B C e^{-A B t} + 1}
\]

• 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

<table>
<thead>
<tr>
<th>System</th>
<th>MSLOC</th>
<th>Known Defects (1000s)</th>
<th>( D_{KD} ) (/Kloc)</th>
<th>Known Vulnerabilities</th>
<th>( V_{KD} ) (/Kloc)</th>
<th>Ratio ( V_{KD}/D_{KD} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win 95</td>
<td>15</td>
<td>5</td>
<td>0.33</td>
<td>46</td>
<td>0.0031</td>
<td>0.92%</td>
</tr>
<tr>
<td>NT 4.0 server</td>
<td>16</td>
<td>10</td>
<td>0.625</td>
<td>162</td>
<td>0.0101</td>
<td>1.62%</td>
</tr>
<tr>
<td>Win 98</td>
<td>18</td>
<td>10</td>
<td>0.556</td>
<td>84</td>
<td>0.0047</td>
<td>0.84%</td>
</tr>
<tr>
<td>Win 2000</td>
<td>35</td>
<td>63</td>
<td>1.8</td>
<td>508</td>
<td>0.0145</td>
<td>0.81%</td>
</tr>
<tr>
<td>Win XP</td>
<td>40</td>
<td>106.5*</td>
<td>2.66*</td>
<td>728</td>
<td>0.0182</td>
<td>0.68%*</td>
</tr>
<tr>
<td>Apache HTTP 2006</td>
<td>227 (Unix)</td>
<td>4148</td>
<td>18.27</td>
<td>96</td>
<td>0.423</td>
<td>2.32%</td>
</tr>
<tr>
<td>Firefox</td>
<td>2.5</td>
<td>24,027</td>
<td>9.61</td>
<td>134</td>
<td>0.0536</td>
<td>0.557%</td>
</tr>
</tbody>
</table>

*MS Thesis Woo, 2006*
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.
• Classification of Time-based VDMs.
### Table of models and their equations

<table>
<thead>
<tr>
<th>Model</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NHPP Power-law</strong> (Movahedi et al., 2018)</td>
<td>$\Omega(t) = (\beta^{-\alpha}) \cdot t^\alpha$</td>
</tr>
<tr>
<td><strong>Gamma-based VDM</strong> (Joh and Malaiya, 2014)</td>
<td>$\Omega(t_0) = \int_{t=0}^{\infty} \frac{\gamma}{\Gamma(\alpha)} \beta^\alpha t^{\alpha-1} e^{-\frac{\beta}{t}} , dt$</td>
</tr>
<tr>
<td><strong>Weibull-based VDM</strong> (Kim et al., 2007)</td>
<td>$\Omega(t) = \gamma \left{ 1 - e^{-\left( \frac{t}{\beta} \right)^\alpha} \right}$</td>
</tr>
<tr>
<td><strong>AML VDM</strong> (Alhazmi and Malaiya, 2005)</td>
<td>$\Omega(t) = \frac{\beta \gamma}{\beta \gamma + 1}$</td>
</tr>
<tr>
<td><strong>Normal-based VDM</strong> (Joh and Malaiya, 2014)</td>
<td>$\Omega(t) = \frac{\gamma}{1 + e^{-\frac{t}{\mu}}}$</td>
</tr>
<tr>
<td><strong>Rescorla Exponential (RE)</strong> (Rescorla, Jan. 2005)</td>
<td>$\Omega(t) = \gamma \left( 1 - e^{-\lambda t} \right)$</td>
</tr>
<tr>
<td><strong>Rescorla Quadratic (RQ)</strong> (Rescorla, Jan. 2005)</td>
<td>$\Omega(t) = \frac{4t^2}{2} + Bt$</td>
</tr>
<tr>
<td><strong>Younis Folded (YF)</strong> (Younis et al., 2011)</td>
<td>$\Omega(t) = \frac{\gamma}{2} \left{ \text{erf} \left( \frac{t-t_0}{\sqrt{2\sigma}} \right) + \text{erf} \left( \frac{t+t_0}{\sqrt{2\sigma}} \right) \right}$</td>
</tr>
<tr>
<td><strong>Linear Model (LM)</strong> (Alhazmi and Malaiya, 2006)</td>
<td>$\Omega(t) = At + B$</td>
</tr>
</tbody>
</table>

---

Vulnerability Discovery Models

Seasonality in Vulnerability Discovery
Seasonality in Vulnerability Discovery

- **Vulnerability Discovery Model (VDM):**
  - a probabilistic methods for modeling the discovery of software vulnerabilities \cite{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
- Significance
  - Enhance VDMs’ predicting ability
- Annual and Weekly seasonality
Annual: Prevalence in Month

Vulnerabilities Disclosed

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

Percentage of Vuln. for Month

- Jan: Win NT, IIS, IE
- Feb: Win NT, IIS, IE
- Mar: Win NT, IIS, IE
- Apr: Win NT, IIS, IE
- May: Win NT, IIS, IE
- Jun: Win NT, IIS, IE
- Jul: Win NT, IIS, IE
- Aug: Win NT, IIS, IE
- Sep: Win NT, IIS, IE
- Oct: Win NT, IIS, IE
- Nov: Win NT, IIS, IE
- Dec: Win NT, IIS, IE

Legend:
- Win NT
- IIS
- Internet Explorer
Seasonal Index

- **Seasonal index**: measures how much the average for a particular period tends to be above (or below) the expected value.

- **H₀**: 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.

---

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]:
  
  \[
  r_k = \frac{\sum_{t=b}^{n} (z_t - \bar{z})(z_{t+k} - \bar{z})}{\sum_{t=b}^{n} (z_t - \bar{z})^2}
  \]
  
  where
- 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

---

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.
Why seasonality?

Weekly Seasonality

Figure 1. Run charts for unpatched critical vulnerabilities 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.

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”

Quantitative Security

Colorado State University
Yashwant K Malaiya
CS 559
Multi-version Systems

CSU Cybersecurity Center
Computer Science Dept
Vulnerability Discovery in Multi-Version Software Systems

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

Software Evolution: Apache & Mysql

Modification: Apache 43%, Mysql 31%

Some vulnerabilities are in added code, many are inherited from precious versions.
• Observation
  – Vulnerability increases after saturation in AML modeling

• Accounting for Superposition Effect
  – Shared components between several versions of software
Multi-version Vulnerability Discovery

![Multiple Software Vulnerability Discovery Trend](image)

\[ \Omega(t) = \frac{B}{B e^{-\beta t} + 1 - \alpha + B' e^{-\beta(t-\delta)}} \]

<table>
<thead>
<tr>
<th></th>
<th>Previous Version</th>
<th>Next Version</th>
<th>Shared Code Ratio ((\alpha))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>1.3.24 (3-21-2002)</td>
<td>2.0.35 (4-6-2002)</td>
<td>20.16%</td>
</tr>
<tr>
<td>Mysql</td>
<td>4.1.1 (12-1-2003)</td>
<td>5.0.0 (12-22-2003)</td>
<td>83.52%</td>
</tr>
</tbody>
</table>
One vs Two Humps

One-humped Vulnerability Discovery Model

Superposition affect
• 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: Total vulnerabilities, Microsoft products
Long Term Trends

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

- Size evolution: Linus kernel
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
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