CSE 227
Computer Security

Spring 2010

Software Vulnerabilities II

Stefan Savage

(many slides courtesy Ozment/Schecter)
Different side of the problem

• Given: vulnerabilities are introduced to software

• Rescorla
  – Is it worth putting in effort to find and fix them?

• Ozment/Schecter
  – Is software getting appreciably more secure over time?
Value of vulnerability finding

• Social benefits of vulnerability finding
  – Pressure vendors into improving code
  – Find and fix bugs before bad guys exploit them
  – Improve security of existing software

• These are assertions… can we test any of them?
Vulnerability life cycle

1. Vulnerability introduced
2. Discovery (good guys or bad guys)
3. Disclosure
4. Exploitation
5. Fix

Order can vary: 12345, 12435 both common today

Value of bug finding is reducing time of exploitation (try to ensure that t4-t5 is positive and large)

Aside: impact of disclosure/fix on exploitation…
Ekr’s idea

• From a value standpoint the key issue is the likelihood of rediscovery
  – Aside: reflected in whether internally discovered vulnerabilities are put in public patches

• Model: assume vulnerability discovery is stochastic
  – If we know discovery rate the we can infer rediscovery probability
  – Idea: try to infer discovery rate (and number of vulnerabilities) from empirical data via model fitting

• ICAT (now NVD) data
  – Vulnerability disclosures indexed by CVE
  – Focused on software with at least 20 vulnerabilities
  – Tricky: dirty data and hard to control for all factors
Figure 8 Vulnerability discovery rate by programs
Overall results

Looks at data from three perspectives

1. Software:
   • Four operating systems
   • Linear and exponential models do not fit

2. Vulnerability age cohorts
   • Four years: 1997-2000, inclusive
   • Only 1999 shows trend

3. All vulnerabilities
   • Half life of 2.5 years
Rescorla concludes

- Vulnerability finding does not improve software quality appreciably
  - No significant reduction in finding rate, hence still jizillions of bugs
- Likelihood of rediscovery is therefore small

- Claim: finding vulnerabilities is thus unlikely to be a good idea (from a maximizing social good standpoint)
Milk or Wine

• Ozment and Schechter concerned that NVD data is noisy
  – Not comprehensive
  – Doesn’t capture vulnerability introduction (time to discovery)
  – Inaccurate discovery date

• Instead focus on single system (OpenBSD)
  – Revision control history for all code
  – Gather comprehensive vulnerability data
Key question:
Are rates of vulnerability reports decreasing…
...or not?
The OpenBSD data set:
140 vulnerabilities reported in 7.5 years

An event is a vulnerability if:

1. It is listed in the OpenBSD security page, Bugtraq, OSVDB, NVD (ICAT), or ISS X-Force

2. There is no vulnerability of the same type reported that week
Study period

- OpenBSD
  - 7.5 years
  - 15 releases
  - 05/1998 to 11/2005

- NetBSD
  - Version between each version

- Undocumented vulnerability fixes

- Source files restructured

- 6 months between each version
Vulnerability lifecycle

<table>
<thead>
<tr>
<th>Birth=</th>
<th>02/99</th>
<th>Introduced</th>
<th>cvs commit -m “new buffer code”</th>
</tr>
</thead>
<tbody>
<tr>
<td>05/99</td>
<td>Released</td>
<td>version 2.5 available for download</td>
<td></td>
</tr>
<tr>
<td>05/02</td>
<td>Published</td>
<td>mail bugtraq, openbsd</td>
<td></td>
</tr>
<tr>
<td>02/02</td>
<td>Repaired</td>
<td>cvs commit -m “vulnerability fix”</td>
<td></td>
</tr>
</tbody>
</table>
Lines of code vs. number vulnerabilities

No correlation
Vulnerability densities

• Wide range of vulnerability densities
  – Inflated LOC counts

• Version 2.4
  – Internet Key Exchange (2)
  – OpenSSL (3)

• Compare with conventional wisdom
  – Vulnerabilities: < 0.033 / 1K LOC
  – Defects: 8-12 / 1K LOC

<table>
<thead>
<tr>
<th>Vers.</th>
<th># vulns</th>
<th>1M LOC</th>
<th># vulns</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3</td>
<td>87</td>
<td>10.14</td>
<td>8.58</td>
</tr>
<tr>
<td>2.4</td>
<td>14</td>
<td>0.42</td>
<td>33.04</td>
</tr>
<tr>
<td>2.5</td>
<td>6</td>
<td>0.27</td>
<td>21.84</td>
</tr>
<tr>
<td>2.6</td>
<td>9</td>
<td>1.05</td>
<td>8.55</td>
</tr>
<tr>
<td>2.7</td>
<td>7</td>
<td>0.77</td>
<td>9.10</td>
</tr>
<tr>
<td>2.8</td>
<td>0</td>
<td>0.40</td>
<td>0.00</td>
</tr>
<tr>
<td>2.9</td>
<td>4</td>
<td>2.23</td>
<td>1.79</td>
</tr>
<tr>
<td>3.0</td>
<td>5</td>
<td>0.62</td>
<td>8.00</td>
</tr>
<tr>
<td>3.1</td>
<td>2</td>
<td>0.81</td>
<td>2.46</td>
</tr>
<tr>
<td>3.2</td>
<td>2</td>
<td>0.33</td>
<td>6.03</td>
</tr>
<tr>
<td>3.3</td>
<td>2</td>
<td>0.32</td>
<td>6.18</td>
</tr>
<tr>
<td>3.4</td>
<td>0</td>
<td>0.83</td>
<td>0.00</td>
</tr>
<tr>
<td>3.5</td>
<td>2</td>
<td>1.44</td>
<td>1.39</td>
</tr>
<tr>
<td>3.6</td>
<td>0</td>
<td>0.74</td>
<td>0.00</td>
</tr>
<tr>
<td>3.7</td>
<td>0</td>
<td>0.91</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Total 140 21.45 6.53
Vulnerabilities’ time of death

Version current

Vulnerability deaths

< 2.3
2.4
2.5
2.6
2.7
2.8
2.9
3.0
3.1
3.2
3.3
3.4
3.5
3.6
3.7

25
20
15
10
5
0
Vulnerabilities’ time of death

Birth version

- 3.5
- 3.3
- 3.2
- 3.1
- 2.9
- 2.7
- 2.6
- 2.5
- 2.4
- 2.3

Vulnerability deaths

Version current

- ≤2.3
- 2.4
- 2.5
- 2.6
- 2.7
- 2.8
- 2.9
- 3.0
- 3.1
- 3.2
- 3.3
- 3.4
- 3.5
- 3.6
- 3.7
Why so many foundational vulnerabilities?
Source code composition over time

Lines of code contributed by each earlier version

Composite version

Contributor version

Millions

25
20
15
10
5
0

2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1 3.2 3.3 3.4 3.5 3.6 3.7
So we’ll analyze the foundation version (2.3)

Not enough data for analysis
Foundational vulnerabilities:
Median lifetime lower bound

Percent Remaining

Age in years

961 days
Raw data: time between vulnerability reports

- Mean: 29.1
- Median: 18
- Standard Deviation (σ): 29.1
- Range: 1 - 126
Eight slices of raw data

Mean: 29.1
Median: 18
$\sigma$: 29.1
Range: 1 - 126
Eight slices of raw data

Mean: 29.1
Median: 18
\[ \sigma \] (Standard Deviation): 29.1
Range: 1 - 126
Number of vulnerabilities per study eighth

Confidence intervals derived from a normal approximation of a homogenous Poisson process

Study period
(Each period of length 342 days)
Two slices of raw data

Mean: 29.1
Median: 18
\( \sigma \) 29.1
Range: 1 - 126

Days Since Study Began
Number of vulnerabilities per study half

Confidence intervals derived from a normal approximation of a homogenous Poisson process

Study period
(Each period of length 1369 days)
Laplace test

H<sub>1</sub>: Increasing rate

H<sub>0</sub>: No trend

H<sub>2</sub>: Decreasing rate

Assumes vulnerability reporting is a non-homogenous Poisson process
Laplace test results

H₁: Increasing rate

H₀: No trend

H₂: Decreasing rate

Age of vulnerability (Years)
Conclusion by authors

Code security of OpenBSD 2.3 is improving with age

Estimate 42 vulnerabilities remaining in foundation set
Limitations

Cannot normalize for changes in:

• Uniqueness of OpenBSD dataset
• Right censoring
• Changes in incentives related to vulnerability funding
  – Number of vulnerability hunters (and $ paid)
  – Effort expended by vulnerability hunters
  – Vulnerability hunting fads
    • Image file processing
    • Microsoft
• Impact of focused tools
  – E.g., file format fuzzers
• Discovered but undisclosed vulnerabilities
• Others?
For next time

- Automating exploit creation and vulnerability defense