Zero Days, Thousands of Nights
The life and times of zero-day vulnerabilities and their exploits

Lillian Ablon
lablon@rand.org
@lilyablon
Are zero-day vulnerabilities a zero-sum game?

- Zero-day vulnerabilities can be very useful to those testing defenses or planning offensive operations
  - They can also lead to unsecure platforms and increase risk

Retain or disclose?
Retain or disclose?

Should a government keep zero-days secret?

Should a government disclose zero-days?
The decision calculus is complicated:
The decision calculus is complicated: there are many equities to consider

- Defense
- Intelligence, law enforcement, and operational
- Commercial
- International partnership
The decision calculus is complicated: there are many variables in play

- The product that the vulnerability is in
- The threat actor that might take advantage of the vulnerability
- The use of the vulnerability in operations
- The vulnerability itself
- Other information

These variables are a few of those that are examined as part of the U.S. Vulnerabilities Equities Process.
The decision calculus is complicated: there are many variables in play

- The product that the vulnerability is in
- The threat actor that might take advantage of the vulnerability
- The use of the vulnerability in operations
- The vulnerability itself
- Other information

These variables are a few of those that are examined as part of the U.S. Vulnerabilities Equities Process
We focus on characteristics of the vulnerabilities

• Challenge: publicly available information about zero-days is sparse

• Goal: create some baseline metrics on the characteristics of zero-day vulnerabilities, using actual zero-day data, in order to help inform policy and technical discussions
We focus on characteristics of the vulnerabilities

<table>
<thead>
<tr>
<th>Life Status</th>
<th>Longevity</th>
<th>Collision Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Who knows about the vulnerability?</td>
<td>How long will the vulnerability remain publicly unknown?</td>
<td>How many vulns get independently rediscovered and publicly disclosed?</td>
</tr>
</tbody>
</table>
• Research Focus
• Quick Dive into the Data
• Analysis & Findings
• Implications & Recommendations
Various groups search for vulnerabilities
Private groups consist of ‘good’ and ‘bad’ actors

- Companies / vendors looking for zero-day vulnerabilities in their own products and products of their customers
- Bug Hunters looking for zero-day vulnerabilities, often for bug bounty payouts
- Zero-day subscription feed businesses
- Other organizations like Project Zero

Adversaries of US, Malicious Actors
Sometimes different groups find the same vuln.
Disclosure affects each camp differently

Vulnerabilities known to both US and THEM disclosure by US may strengthen our defensive posture.
Disclosure affects each camp differently

Vulnerabilities known *only* to US, and not to THEM:

disclosure by US may hinder our offensive posture
Disclosure affects each camp differently
Large overlap: We’re vulnerable! Disclose all!
Small overlap: We’re (mostly) secure; retain!
What about the overlap between us and them?
What about the overlap between us and them?

BUSBY is our proxy

Vulnerabilities known to BUSBY; not in Public Knowledge

Vulnerabilities in the private-public overlap between BUSBY and Public Knowledge
BUSBY finds zero-day vulnerabilities, and develops exploits for them

14
Year span
(2002-2016)

207
Vulnerabilities
and their exploits

64
Vendors

Data consists of information about vulnerability class, source code type, exploit class type, vendor, product, exploit developer, and various dates (vulnerability discovery, exploit developed)
Data stats: three main types of vulnerabilities

- Memory Corruption: 110
- Memory Mismanagement: 41
- Logic: 67
Vulnerability Sub-Type: Memory Corruption

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSS Overflow</td>
<td>1</td>
</tr>
<tr>
<td>Data overflow</td>
<td>1</td>
</tr>
<tr>
<td>Heap Overflow</td>
<td>58</td>
</tr>
<tr>
<td>Integer overflow</td>
<td>2</td>
</tr>
<tr>
<td>Integer truncation</td>
<td>2</td>
</tr>
<tr>
<td>Stack overflow</td>
<td>40</td>
</tr>
<tr>
<td>Heap + Stack</td>
<td>1</td>
</tr>
<tr>
<td>Heap + Integer</td>
<td>1</td>
</tr>
</tbody>
</table>
## Vulnerability Sub-Type: Memory Mismanagement

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remap memory</td>
<td>1</td>
</tr>
<tr>
<td>Information leak</td>
<td>4</td>
</tr>
<tr>
<td>Integer mismanagement</td>
<td>1</td>
</tr>
<tr>
<td>Invalid pointer dereference</td>
<td>2</td>
</tr>
<tr>
<td>Name validation</td>
<td>1</td>
</tr>
<tr>
<td>Null dereference</td>
<td>12</td>
</tr>
<tr>
<td>Out of bounds write</td>
<td>1</td>
</tr>
<tr>
<td>Privilege escalation</td>
<td>2</td>
</tr>
<tr>
<td>Reference count + object mismanagement</td>
<td>1</td>
</tr>
<tr>
<td>Type confusion + object mismanagement</td>
<td>1</td>
</tr>
<tr>
<td>Unsecure environment variables</td>
<td>1</td>
</tr>
<tr>
<td>Use after free</td>
<td>2</td>
</tr>
<tr>
<td>Use unverified supply pointer value</td>
<td>2</td>
</tr>
</tbody>
</table>
## Vulnerability Sub-Type: Logic

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Misuse</td>
<td>3</td>
</tr>
<tr>
<td>Authentication Bypass</td>
<td>5</td>
</tr>
<tr>
<td>Auto execution</td>
<td>1</td>
</tr>
<tr>
<td>Bypass</td>
<td>1</td>
</tr>
<tr>
<td>Call-gate mismanagement</td>
<td>2</td>
</tr>
<tr>
<td>Command injection</td>
<td>3</td>
</tr>
<tr>
<td>Design misuse</td>
<td>1</td>
</tr>
<tr>
<td>Directory traversal; input validation</td>
<td>1</td>
</tr>
<tr>
<td>DNS Cache poisoning</td>
<td>1</td>
</tr>
<tr>
<td>Environment insertion</td>
<td>1</td>
</tr>
<tr>
<td>Executable file upload</td>
<td>1</td>
</tr>
<tr>
<td>File normalization error</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>File read primitive</td>
<td>2</td>
</tr>
<tr>
<td>IO control based on write primitives</td>
<td>1</td>
</tr>
<tr>
<td>Object injection / deserialization</td>
<td>4</td>
</tr>
<tr>
<td>Permissions on kernel device</td>
<td>1</td>
</tr>
<tr>
<td>Privilege issues: file read (1); mismanagement (2); spoofing (1)</td>
<td>4</td>
</tr>
<tr>
<td>Race condition</td>
<td>20</td>
</tr>
<tr>
<td>Reference count</td>
<td>3</td>
</tr>
<tr>
<td>Register / memory mismanagement</td>
<td>1</td>
</tr>
<tr>
<td>Remote code injection</td>
<td>1</td>
</tr>
<tr>
<td>SQLi</td>
<td>1</td>
</tr>
<tr>
<td>XSS</td>
<td>1</td>
</tr>
</tbody>
</table>
Data stats: number of vulnerabilities per source code type

- Closed: 123
- Open: 74
- Mix or N/A: 10
### Data stats: number of vulnerabilities found and exploited, by vendor

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Microsoft</th>
<th>Linux</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft</td>
<td>55</td>
<td>39</td>
<td>88</td>
</tr>
<tr>
<td>Apple</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUN/Oracle</td>
<td>11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 64 vendors total
- Others include: Mozilla, LinkSys, Google, Adobe, etc.
## Data stats: number of exploits developed per exploit class type

<table>
<thead>
<tr>
<th>Type</th>
<th>Exploits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>76</td>
</tr>
<tr>
<td>Client-side</td>
<td>25</td>
</tr>
<tr>
<td>Remote</td>
<td>71</td>
</tr>
</tbody>
</table>
Some other observations about the data

- 4% of the vulnerabilities in the dataset were purchased from an outside 3rd party
- Not all vulnerabilities were exploited
- CVEs do not always provide accurate and complete information about the severity of a vulnerability
- Exploitability is fluid, and can shift over time
- Virtual isolation (hypervisors or VMs) and anti-virus are not necessarily viable mitigations
- Other observations (charts) . . .
Exploit development time is relatively short

Over 70% of exploits are developed in a month (31 days) or less
Mitigations have affected exploitability (e.g., heap vs stack overflow)

Mitigations introduced c. 2007 caused a shift in type of buffer overflow exploited
Exploit development career lengths vary

Low hanging fruit may account for a higher number of exploits developed early on.
Exploit development career lengths vary

Each researcher has her/his own unique set of skills and focus
We focus on characteristics of the vulnerabilities.

- **Life Status**
  - ?
  - ?

- **Longevity**
  - Survival Rate
  - Life Expectancy

- **Collision Rate**
There are some caveats to our research

- Results from our research can be generalized only to similar datasets
- We are comparing private data to public data (ideal would be to compare multiple private datasets)
Life Status

Research Question: What are various “life stages” a zero-day vulnerability can be in?

Metric: What proportion of zero-day vulnerabilities are:
  • Alive (publicly unknown / blue)
  • Dead (publicly known / teal & green)
  • Somewhere in between
Alive and dead are numbered about the same

There is more granularity to a vulnerability being either alive or dead
We found more granularity in life stages stages.

Labeling a vulnerability as either alive or dead is misleading and too simplistic.
About 1 in 6 of the alive are immortal

Labeling a vulnerability as either alive or dead is misleading and too simplistic.
Labeling a vulnerability as either alive or dead is misleading and too simplistic.
Code revisions created a bunch of code refactored “zombies”

Labeling a vulnerability as either alive or dead is misleading and too simplistic
Research Question: How long will a zero-day vulnerability remain undiscovered and undisclosed to the public?

Metrics:

• What is a short and long life for a zero-day vulnerability?
• What is the average life expectancy of a zero-day vulnerability and its exploit?
We borrowed a methodology from life insurers

- We do not know what is going to happen to those vulnerabilities that are still currently alive
  - Calculating short life, long life, and average lifetimes requires taking into account alive vulnerabilities

- Kaplan-Meier analysis estimates the probability of surviving from some event of interest over time
  - Ex: For humans, the probability of someone having a heart attack
  - For vulnerabilities, the probability of dying and becoming publicly known
We plotted the survival probability of our data.
75% lived longer than 1.5 years

25% died before 1.5 years

25% lived longer than 9.5 years
Average life expectancy is nearly 7 years

Average Life Expectancy: **6.9 years**
(Area under exponential curve)
Do certain characteristics indicate a long or short life?

- Vulnerability Type
- Platform/Vendor affected
- Source Code
- Exploit Class
Do certain characteristics indicate a long or short life?

- Vulnerability Type
- Platform/Vendor affected
- Source Code
- Exploit Class
Do certain characteristics indicate a long or short life?

- Vulnerability Type
- Platform/Vendor affected
- Source Code
- Exploit Class
Do certain characteristics indicate a long or short life?

- Vulnerability Type
- Platform/Vendor affected
- Source Code
- Exploit Class
Do certain characteristics indicate a long or short life?

- Vulnerability Type
- Platform/Vendor affected
- Source Code
- Exploit Class
Do certain characteristics indicate a long or short life?

It’s unclear.

More data is needed to refine results.
Does life expectancy or survival probability change over time?

Does not appear so.
Results not statistically significant to indicate a difference year by year.

More data could refine results.
Collision Rate

Research Question: What is the collision rate of zero-day vulnerabilities independently discovered and disclosed in a given time period?

Metric: What percentage of privately known vulnerabilities get independently rediscovered and publicly disclosed in a given time period?
Time interval:
All (14 years)

40%
We examined various time intervals

- Choose a time interval (365 days, 180 days, 90 days)
- Over that time interval, new zero-day vulnerabilities are discovered and retained
- At the end of the time interval, examine how many have been found by others and publicly disclosed (i.e. died)
  - “Throw out” those that have died
  - Keep the ones that are still alive
  - Continue to discover and retain new ones until the end of the next time interval when re-evaluation begins again

Collision rate: median percentage of those that died over all time intervals
Clarity about time intervals is important

Time interval: 365-days (1 year)

5.7%
Clarity about time intervals is important

Time interval: 180-days
(~6 months)

2.8%
Clarity about time intervals is important

Time interval: 90-days

0.87%
<table>
<thead>
<tr>
<th>Time interval:</th>
<th>Collision rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>All (14 years)</td>
<td>40%</td>
</tr>
<tr>
<td>365-days</td>
<td>5.7%</td>
</tr>
<tr>
<td>180-days</td>
<td>2.8%</td>
</tr>
<tr>
<td>90-days</td>
<td>0.87%</td>
</tr>
</tbody>
</table>

Mathematical meaning can be easily manipulated.
We explored several other research paths

- Average life expectancies based on vulnerability characteristic*
- Life expectancy variation based on birth year
- Collision rate variation based on vulnerability characteristic*
- Collision rate and timing for individual vulnerabilities
- Time to develop exploit based on vulnerability characteristic *
- Seasonality of vulnerability research
- Cost of developing an exploit

*No statistical significance found, likely due to limited data

If you have data and would like to collaborate to refine this research, please contact me: lablon@rand.org or @lilyablon
Labeling a zero-day vulnerability as either alive or dead can be misleading and too simplistic.

Zero-day vulnerabilities and their exploits have a rather long average life expectancy.

Time interval examined can significantly change the percentage for likelihood of independent rediscovery.

Implications and recommendations of findings

For those **defensively** focused

- Refine tactical approaches:
  - Analyze previous versions of code that are still in heavy use (e.g., ICS)
  - Harness techniques of how offense finds vulnerabilities
  - Seek better options to detect vulns

- Consider strategic approaches:
  mitigation, containment, accountability, and a robust infrastructure of patching
  - Employ physical isolation
  - Account for software, devices, and removable media
  - Incentivize upgrading to new versions

For those **offensively** focused

- Retain a few vulnerabilities per particular software package
- Consider immortal or code-refactored vulnerabilities for operations
- Regularly revisit vulnerabilities thought to be unexploitable
- Plan for a specific vulnerability only for short-term planning operations; expand to any vulnerability may extend the timeline
Our findings can help inform the retain vs. disclose discussions

Long average lifetimes and relatively low collision rates may indicate that:

vulnerabilities are dense, or vulnerabilities are hard to find
Our findings can help inform the retain vs. disclose discussions

vulnerabilities are dense, or vulnerabilities are hard to find

Pro retention

• The level of protection from disclosing a vulnerability may be modest

• There is a small probability of re-discovery by others

Pro disclosure

• Collision rates for zero-day vulnerabilities are non-zero

• A non-zero probability (no matter how small) that someone else will find the same zero-day vulnerability may be too risky
Zero-days affect many sectors, and raise policy questions

- Should we prioritize national security, or consumer safety and company liability?

- Should software companies be liable for vulnerabilities in their products?

- What is the impact to a business’ risk profile?
Labeling a zero-day vulnerability as either alive or dead can be misleading and too simplistic.

Zero-day vulnerabilities and their exploits have a rather long average life expectancy of 6.9 years.

Time interval examined can significantly change the percentage for likelihood of independent rediscovery, with a collision rate of 5.7% per year.

Thank you!

Lillian Ablon
lablon@rand.org
@LilyAblon