How Long Do Vulnerabilities Live in the Code?
A Large-Scale Empirical Measurement Study on FOSS Vulnerability Lifetimes

Nikolaos Alexopoulos, Manuel Brack, Jan Philipp Wagner, Tim Grube, Max Mühlhäuser
USENIX Security Symposium, Aug. 10th 2022, Boston, MA, USA
What? The vulnerability lifecycle and lifetimes

- **Phase 1**: vuln. introduced
- **Phase 2**: vuln. found
- **Phase 3**: vuln. pub. disclosed
- **Phase 4**: fix available
- **Final Phase**: all hosts patched

**Lifetime of a vulnerability**
Lifetimes in version control systems

VCC: Vulnerability Contributing Commit

Fixing commit

\( \Delta t \)
Why measuring lifetimes is hard

VCC: Vulnerability Contributing Commit

Time

Fixing commit

available

scarce

? seemingly very difficult
VCCFinder [Perl et al. CCS 2015]

CVE-2022-25375

```c
638 | 638 | r = ndis_add_response(params, sizeof(rndis_set_cmplt_type));  Blames[83210e5] += 1

639 | 639 |

640 | 640 | BufLength = le32_to_cpu(buf->InformationBufferLength);
641 | 641 | BufOffset = le32_to_cpu(buf->InformationBufferOffset);
642 | 642 | if ((BufLength > RNDIS_MAX_TOTAL_SIZE) ||
643 | 643 | (BufOffset + 8 >= RNDIS_MAX_TOTAL_SIZE))
644 | 644 | return -EINVAL;
645 | 645 | +

646 | 646 | r = ndis_add_response(params, sizeof(rndis_set_cmplt_type));  Blames[83210e5] += 1

647 | 647 | if (!r)
648 | 648 | return -ENOMEM;
649 | 649 | resp = (rndis_set_cmplt_type *)r->buf;
650 | 650 |

651 | 651 | #ifdef VERBOSE_DEBUG
```

```c
- BufLength = le32_to_cpu(buf->InformationBufferLength);  Blames[aldf4e4] += 1
- BufOffset = le32_to_cpu(buf->InformationBufferOffset);  Blames[aldf4e4] += 1
- Blames[1da177e] += 1
```
VCCFinder [Perl et al. CCS 2015]

Blames[7e27f18] = 1
Blames[83210e5] = 1
**Blames[a1df4e4] = 2** VCC
Blames[1da177e] = 1

Commit with most blames

Listed accuracy (manual check of sample): 96%

We measured (ground-truth data): 40%
How we did it

Key observations:

1. **We do not necessarily need to pinpoint the VCC – we just need to estimate its commit date**
How we did it (cont.)

➔ Use heuristic similar to VCCFinder with **weighted average** over the **blamed commits** (and some improvements introduced in Vuldigger [8])

\[
d_h = d_{\text{ref}} + \frac{1}{\sum_{i=1}^{n} b_i} \sum_{i=1}^{n} b_i (d_i - d_{\text{ref}})
\]

Weights: number of blames of commit i

Date of commit i

Arbitrary reference date

# Heuristic performance

<table>
<thead>
<tr>
<th>Project (CVEs)</th>
<th>Lifetime</th>
<th>Li &amp; Paxson $^7$</th>
<th>our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>ME</td>
<td>St. dev</td>
</tr>
<tr>
<td>Linux (885)</td>
<td>1330.8</td>
<td>-323.7</td>
<td>1033.2</td>
</tr>
<tr>
<td>Chrom. (226)</td>
<td>754.2</td>
<td>-370.3</td>
<td>747.5</td>
</tr>
<tr>
<td>Httpd (60)</td>
<td>1890.2</td>
<td>-599.8</td>
<td>1160.0</td>
</tr>
<tr>
<td>All (1171)</td>
<td>1248.2</td>
<td>-346.8</td>
<td>993.7</td>
</tr>
</tbody>
</table>

How we did it

**Key observations:**

1. **We do not necessarily need to pinpoint the VCC – just estimate commit date**
2. **We do not necessarily care about individual vulnerabilities**

\[ \text{sigma} \sim \frac{1}{\sqrt{N}} \rightarrow \]

- 10 samples 95% CI \( \pm 585 \) days
- 20 samples \( \pm 395 \) days
- 100 samples \( \pm 176 \) days
Validating the heuristic

- Is the heuristic *good enough*? → We need to see how the heuristic performs in tasks similar to what we want to do
Heuristic performance (over time)

Linux: Years with >20 vulnerabilities in ground truth dataset
Heuristic performance (over time)

Heuristic performs well over time and in estimating trends

Linux: Years with >20 vulnerabilities in ground truth dataset
Heuristic performance (distributions)

Heuristic performs well in estimating the distribution of lifetimes
Dataset

- 11 big popular FLOSS projects – multiple sources
- 1.193 CVEs with known VCC (ground truth)
- ~6,000 CVEs with known fixing commit
Results
Results: lifetimes per project

<table>
<thead>
<tr>
<th>Project</th>
<th>Average</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux (kernel)</td>
<td>1732.97</td>
<td>1363.5</td>
</tr>
<tr>
<td>Firefox</td>
<td>1338.58</td>
<td>1082.0</td>
</tr>
<tr>
<td>Chromium</td>
<td>757.59</td>
<td>584.5</td>
</tr>
<tr>
<td>Wireshark</td>
<td>1833.86</td>
<td>1475.0</td>
</tr>
<tr>
<td>Php</td>
<td>2872.40</td>
<td>2676.0</td>
</tr>
<tr>
<td>Ffmpeg</td>
<td>1091.99</td>
<td>845.5</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>2601.91</td>
<td>2509.0</td>
</tr>
<tr>
<td>Httpd</td>
<td>1899.96</td>
<td>1575.5</td>
</tr>
<tr>
<td>Tcpdump</td>
<td>3168.58</td>
<td>3236.0</td>
</tr>
<tr>
<td>Qemu</td>
<td>1743.86</td>
<td>1554.0</td>
</tr>
<tr>
<td>Postgres</td>
<td>2336.56</td>
<td>2140.0</td>
</tr>
<tr>
<td><strong>Average of projects</strong></td>
<td><strong>1943.48</strong></td>
<td><strong>1731.0</strong></td>
</tr>
<tr>
<td><strong>All CVEs</strong></td>
<td><strong>1501.47</strong></td>
<td><strong>1078.0</strong></td>
</tr>
</tbody>
</table>

- Mean: 1943 days → 5.3 years
- Median: 1731 days → 4.7 years
- Median < Mean generally
- Great variations between projects → Do shorter lifetimes mean better security?
Results: the effect of code age

Chromium

Regular code age

Lifetime

year of fixing commit

lifetime in days


0 500 1000 1500 2000
Results: the effect of code age

- Vulnerability lifetime ~ age of the code at time of fix
- Identified metrics:
  - Spread
  - Rate of change of spread
Lifetimes: Implications

- Practical considerations (e.g. LTS duration, tool effectiveness)
- Theoretical insights (e.g. distribution, VDMs)
- Interesting metrics:
  - Spread between average lifetime and code age
  - Rate of change of this spread
- Enables further research
How Long Do Vulnerabilities Live in the Code?
A Large-Scale Empirical Measurement Study on FOSS Vulnerability Lifetimes

Nikolaos Alexopoulos, Manuel Brack, Jan Philipp Wagner, Tim Grube, Max Mühlhäuser

USENIX Security Symposium 2022, Boston, MA, USA

Nikolaos Alexopoulos: alexopoulos@tk.tu-darmstadt.de   @nikanta0
Reproduced Artifact: https://github.com/manuelbrack/VulnerabilityLifetimes/tree/usenix_ae