Understanding the Reproducibility of Crowd-reported Security Vulnerabilities

Dongliang Mu\textsuperscript{12}, Alejandro Cuevas\textsuperscript{2}, Limin Yang\textsuperscript{3}, Hang Hu\textsuperscript{3}, Xinyu Xing\textsuperscript{2}, Bing Mao\textsuperscript{1}, Gang Wang\textsuperscript{3}

1. Nanjing University
2. Pennsylvania State University
3. Virginia Tech
Real World Effects of Security Vulnerabilities

It is infeasible for in-house teams to identify all possible vulnerabilities before a software release
Massive Crowd-reported Vulnerabilities Over Time

The Crowd

- White-hat Hacker
- Security Analysts
- Security Researchers
- Normal User

Vulnerability Reporting Websites

- SecurityFocus.com
- Red Hat Bugzilla
- Google Bounty

CVE¹ Website

¹ Common Vulnerabilities and Exposures
Massive Crowd-reported Vulnerabilities Over Time

Number of vulnerabilities reported to CVE\textsuperscript{1} by year

\textsuperscript{1} Common Vulnerabilities and Exposures
Hi Thuan,

I am unable to reproduce this problem as you reported it. :-(

> binutils was checked out from

How were the binutils configured?

> Its commit is 268ebe95201d2ebdcf68cad9dc67ff6d1e25be9e
> (Fri Nov 18 14:15:12 2016)
Vulnerability Reproduction Can Be Challenging

Nick Clifton 2017-08-09 16:37:39 IST
Hi Zhihua,
I am sorry, but I am unable to reproduce this failure.

Please consider the following:
1. The code was in a . . .

Nick Clifton 2017-06-15 11:14:04 UTC
Hi Aadamski,
I could not reproduce this failure. Please could you check again to see if it is still present? I suspect that one of the recent patches to fix the other problems you detected may have fixed this problem as well.

Comment 4

nozz_ posted a comment.
Hello @Jouko,
Thank you again for your reply.
Unfortunately we did not consider possible to perform any code execution even with your additional information. We tried to reproduce your PoC against our systems but this one is not working mainly because of that environments are not totally similar and our instance is hardened.

rahul 2008-08-08 04:38:16 UTC
That is odd, I couldn't reproduce it, could you please post your httpd.conf in full (and logs with debug on)?
Consequences of Poor Reproducibility

Software vendors

- Poor reproducibility delays the patching of vulnerability

Security Firms

- Poor reproducibility prevents analysts from assessing potential threats to their customers in a timely fashion

Security Researchers

- Poor reproducibility makes it hard to thoroughly evaluate security solutions
Consequences of Poor Reproducibility

Poor reproducibility prevents analysts from assessing potential threats to their customers in a timely fashion.

Poor reproducibility makes it hard to thoroughly evaluate security solutions.

### Research Papers that use public vulnerabilities for evaluation

<table>
<thead>
<tr>
<th>Research Papers</th>
<th># of Vulnerability</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP’2018</td>
<td>9</td>
</tr>
<tr>
<td>Usenix'2017</td>
<td>8</td>
</tr>
<tr>
<td>Usenix'2015</td>
<td>6</td>
</tr>
<tr>
<td>NDSS’2015</td>
<td>7</td>
</tr>
<tr>
<td>Usenix’2015</td>
<td>8</td>
</tr>
<tr>
<td>NDSS’2011</td>
<td>14</td>
</tr>
<tr>
<td>SP’2008</td>
<td>5</td>
</tr>
<tr>
<td>Usenix’2005</td>
<td>4</td>
</tr>
<tr>
<td>Usenix’1998</td>
<td>8</td>
</tr>
</tbody>
</table>

Security Researchers

Poor reproducibility makes it hard to thoroughly evaluate security solutions.
This Work

Q1: How reproducible are public security vulnerability reports?

Q2: What makes vulnerability reproduction difficult?

Q3: How to improve the efficiency of vulnerability reproduction?

We answer three questions by manually reproducing vulnerabilities
We surveyed 48 external security professionals from both academia and industry to examine people’s perceptions towards the vulnerability reports and their usability.
We randomly selected a large collection of reported vulnerabilities

- We focused on Memory Error Vulnerabilities due to their high severity (Average CVSS Score 7.6 > Overall Average CVSS Score 6.2) and significant real-world impact
- We focused on Open Source Linux Software due to debugging and diagnosing capabilities

We collected two datasets including,
- A primary dataset of 291 vulnerabilities with CVE IDs
- A complementary dataset for 77 vulnerabilities without CVE ID

<table>
<thead>
<tr>
<th>CVSS Score</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1 - 3.9</td>
<td>Low</td>
</tr>
<tr>
<td>4.0 - 6.9</td>
<td>Medium</td>
</tr>
<tr>
<td>7.0 - 8.9</td>
<td>High</td>
</tr>
<tr>
<td>9.0 - 10.0</td>
<td>Critical</td>
</tr>
</tbody>
</table>
We collect vulnerability reports by crawling the references listed in the CVE website.

6044 vulnerability reports in total
We collect vulnerability reports by crawling the references listed in the CVE website.

- **6044 vulnerability reports** in total.

### Vulnerability Report Dataset (cont.)

#### CVE-ID: CVE-2008-5314

**Learn more**

**CVSS Severity:** Medium

**Description**

Stack consumption vulnerability in libclamav file, related to the cli_check_jpeg_expand function.

**References**

- EXPLOIT-DB:7330
- URL:https://www.exploit-db.com/7330
- MLIST:[clamav-announce] 20081105 C:
- [clamav-announce] URL:http://lurker.clamav.net/msg/001b
- [oss-security] 20081201 C:
- URL:http://www.openwall.com/lists/oss-security/2008/12/01

---

### Top 5 source websites in our dataset

1. **EXPLOIT DATABASE**
2. **SecurityFocus.com**
3. **Red Hat Bugzilla**
4. **Openwall Security tracker**

---

Vulnerability reports...
The Analyst Team

• We formed a team of 5 security analysts to carry out our experiments

Security Analysts

- In-depth knowledge of memory error vulnerabilities
- First-hand experience analyzing vulnerabilities, writing exploits, and developing patches
- Rich Catch-The-Flag experience, and have discovered and reported over 20 new vulnerabilities to CVE website
Reproduction Workflow

1. Read Reports
2. Set up Environment
3. Install & Config Software
4. Trigger Vulnerability
5. Verify Vulnerability
6. Security Analysts
7. Default Setting for missing information

- Vulnerable Version
- Operating System
- Software Installation
- Software Configuration
- Proof-of-Concept File
- Trigger Method
- Vulnerability Verification
Reproduction Workflow (cont.)

- Set up the operating system for vulnerable software analysis

<table>
<thead>
<tr>
<th>Information</th>
<th>Default Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>A Linux system that was released in (or slightly before) the year when the vulnerability was reported</td>
</tr>
</tbody>
</table>

- Vulnerable Version
- Operating System
- Software Installation
- Software Configuration
- Proof-of-Concept File
- Trigger Method
- Vulnerability Verification
Reproduction Workflow (cont.)

- Compile vulnerable software with the compilation options
- Install vulnerable software with the configuration options

<table>
<thead>
<tr>
<th>Building System</th>
<th>Default Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>automake</td>
<td>make; make install</td>
</tr>
<tr>
<td>autoconf &amp; automake</td>
<td>./configure; make; make install</td>
</tr>
<tr>
<td>cmake</td>
<td>mkdir build; cd build; cmake ..;/ make; make install</td>
</tr>
</tbody>
</table>

- Vulnerable Version
- Operating System
- Software Installation
- Software Configuration
- Proof-of-Concept File
- Trigger Method
- Vulnerability Verification
Reproduction Workflow (cont.)

- Trigger the vulnerability by using the Proof-of-Concept File

<table>
<thead>
<tr>
<th>Type of PoC</th>
<th>Default Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shell commands</td>
<td>Run the commands with the default shell</td>
</tr>
<tr>
<td>Script program (e.g., python)</td>
<td>Run the script with the appropriate interpreter</td>
</tr>
<tr>
<td>C/C++ code</td>
<td>Compile code with default options and run it</td>
</tr>
<tr>
<td>A long string</td>
<td>Directly input the string to the vulnerable program</td>
</tr>
<tr>
<td>A malformed file (e.g., jpeg)</td>
<td>Input the file to the vulnerable program</td>
</tr>
</tbody>
</table>

- Vulnerable Version
- Operating System
- Software Installation
- Software Configuration
- Proof-of-Concept File
- Trigger Method
- Vulnerability Verification
Reproduction Workflow (cont.)

- Verify the vulnerability with expected program behavior

<table>
<thead>
<tr>
<th>Information</th>
<th>Default Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vulnerability Verification</td>
<td>Unexpected program termination (or program “crash”)</td>
</tr>
</tbody>
</table>

- Vulnerable Version
- Operating System
- Software Installation
- Software Configuration
- Proof-of-Concept File
- Trigger Method
- **Vulnerability Verification**
Reproduction Experiment: Controlled Information Source

One of Top 5 Source Websites

- SecurityFocus
- Exploit DataBase
- Redhat Bugzilla
- SecurityTracker
- OpenWall

Single-source

- SecurityFocus
- Exploit DataBase
- OpenWall
- SecurityTracker
- Redhat Bugzilla

Combined-top5

- SecurityFocus
- Exploit DataBase
- OpenWall
- SecurityTracker
- Redhat Bugzilla

Combined-all

- SecurityFocus
- Exploit DataBase
- OpenWall
- SecurityTracker
- Redhat Bugzilla

...
Roadmap

• Methodology
• Findings
• Suggestions
• Conclusion
Finding 1: Vulnerability Is Difficult to Reproduce

<table>
<thead>
<tr>
<th>Information Source</th>
<th>CVE Reproduction (N=291)</th>
<th>Non-CVE Reproduction (N=77)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># of Case</td>
<td># of Success</td>
</tr>
<tr>
<td>SecurityFocus</td>
<td>256</td>
<td>32</td>
</tr>
<tr>
<td>Redhat Bugzilla</td>
<td>195</td>
<td>19</td>
</tr>
<tr>
<td>ExploitDB</td>
<td>156</td>
<td>46</td>
</tr>
<tr>
<td>OpenWall</td>
<td>153</td>
<td>67</td>
</tr>
<tr>
<td>SecurityTracker</td>
<td>89</td>
<td>4</td>
</tr>
<tr>
<td>Combined-top5</td>
<td>287</td>
<td>126</td>
</tr>
<tr>
<td>Combined-all</td>
<td>291</td>
<td>182</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The single-source returns a low success rate.

“Combined-top5” has clearly improved the success rate.

The success rate is improved to 62.5% by “Combined-all”.

Finding 2: Key Factors Make Reproduction Difficult

Intensive manual debugging takes another 2,000 man-hours to finish, about 13 hours for each case.

<table>
<thead>
<tr>
<th>Reproduction State After Manual Debugging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success in Combined-all</td>
</tr>
<tr>
<td>Reproduced by Manual Debugging</td>
</tr>
<tr>
<td>Failure after Manual Effort</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Report Information</th>
<th># of vulnerabilities addressed by Manual Debugging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger Method</td>
<td>74</td>
</tr>
<tr>
<td>Software Installation</td>
<td>43</td>
</tr>
<tr>
<td>PoC File</td>
<td>38</td>
</tr>
<tr>
<td>Software Configuration</td>
<td>6</td>
</tr>
<tr>
<td>OS information</td>
<td>4</td>
</tr>
<tr>
<td>Software version</td>
<td>1</td>
</tr>
<tr>
<td>Vulnerability Verification</td>
<td>0</td>
</tr>
</tbody>
</table>
Finding 3: Useful Tips for Information Recovery

For 74 cases that failed on trigger method, we recovered 68 cases by reading other similar vulnerability reports.
Roadmap

• Methodology
• Findings
• Suggestions
• Conclusion
Our Ideas of Making Vulnerability Reproduction Easier

1. Standardize Vulnerability Reports
2. Develop Useful Automated Tools to Collection Information
3. Automate the Vulnerability Reproduction

Manually generating standardized reports is really time-consuming

With standardized reports, it's a waste of resource if we still reproduce vulnerability entirely by manual efforts
Conclusion

Vulnerability reproduction is difficult and requires extensive manual efforts

A crowdsourcing approach could increase the reproducibility

Apart from manual debugging based on experience, Internet-scale crowdsourcing and some heuristics could help recover missing information

There is an urgent need to automate vulnerability reproduction and overhaul current vulnerability reporting systems
Data Sharing

- **DataSet**: [https://vulnreproduction.github.io/](https://vulnreproduction.github.io/) (12 Virtual Machine Images)
- **Github Repo**: [https://github.com/VulnReproduction/LinuxFlaw](https://github.com/VulnReproduction/LinuxFlaw)

We provide 300+ Reproducible Vulnerabilities in above Repo.

For each vulnerability, we have:

- Fully-tested Proof-of-Concept
- Pre-configured virtual machine or Docker Image
- Detailed instructions on how to reproduce the vulnerability
- Structured information fields (in HTML and JSON)

Name: Dongliang Mu
Homepage: [http://mudongliang.me/about/](http://mudongliang.me/about/)
Email: dzm77@ist.psu.edu
## References

<table>
<thead>
<tr>
<th>Year</th>
<th>Paper Title</th>
<th># of Vulnerability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usenix’2005</td>
<td>Non-control-data attacks are realistic threats</td>
<td>4</td>
</tr>
<tr>
<td>SP’2008</td>
<td>Preventing memory error exploits with wit</td>
<td>5</td>
</tr>
<tr>
<td>Usenix’2015</td>
<td>Control-flow bending: on the effectiveness of control-flow integrity</td>
<td>6</td>
</tr>
<tr>
<td>NDSS’2015</td>
<td>Preventing Use-after-free with Dangling Pointers Nullification</td>
<td>7</td>
</tr>
<tr>
<td>Usenix’1998</td>
<td>StackGuard : automatic adaptive detection and prevention of buffer-overflow attacks</td>
<td>8</td>
</tr>
<tr>
<td>Usenix’2017</td>
<td>Towards efficient heap overflow discovery</td>
<td>8</td>
</tr>
<tr>
<td>Usenix’2015</td>
<td>Automatic Generation of Data-Oriented Exploits</td>
<td>8</td>
</tr>
<tr>
<td>SP’2018</td>
<td>Data-oriented programming : On the Expressiveness of Non-Control Data Attacks</td>
<td>9</td>
</tr>
<tr>
<td>NDSS’2011</td>
<td>AEG: Automatic exploit generation</td>
<td>14</td>
</tr>
</tbody>
</table>