V1SCAN: Discovering 1-day Vulnerabilities in Reused C/C++ Open-source Software Components Using Code Classification Techniques

Seunghoon Woo, Eunjin Choi, Heejo Lee, Hakjoo Oh

Korea University

USENIX Security 2023
Motivation

Open-source software (OSS), a driving force behind innovative software development

• Unmanaged OSS reuse can cause security threats
Motivation

Open-source software (OSS), a driving force behind innovative software development

• Unmanaged OSS reuse can cause security threats

Q1. Do third-party OSS components contain vulnerabilities?
Motivation

Open-source software (OSS), a driving force behind innovative software development

- Unmanaged OSS reuse can cause security threats

Q2. Do vulnerabilities in third-party components exist in the target program?
Motivation

Two main approaches for 1-day vulnerability discovery in C/C++ software

(1) Version-based approach

- Detecting vulnerabilities based on the version information of reused third-party OSS components
e.g., CENTRIS [ICSE ’21], OSSFP [ICSE ’23]

(2) Code-based approach

- Identifying codes syntactically or semantically similar to vulnerable code
e.g., VUDDY [S&P ’17], MVP [SECURITY ’20], MOVERY [SECURITY ’22]
Challenge

Addressing modified OSS reuse

• Existing version-based approaches for C/C++ software
  ▪ Producing false positives
  ▪ Unused or resolved vulnerabilities cannot be addressed effectively

  ![Diagram showing ReactOS Reuse Libtirpc (v0.1.11)]

  • CVE-2017-8779
  • CVE-2018-14621
  • CVE-2018-14622

• Existing code-based approaches for C/C++ software
  ▪ Producing false negatives
  ▪ Vulnerabilities in modified code cannot be detected effectively
Challenge

Addressing **modified OSS reuse**

- Existing version-based approaches for C/C++ software
  - Producing **false positives**
  - Unused or resolved vulnerabilities cannot be addressed effectively

- Existing code-based approaches for C/C++ software
  - Producing **false negatives**
  - Vulnerabilities in modified code cannot be detected effectively
V1SCAN: Discovering 1-day Vulnerabilities in Reused C/C++ Open-source Software Components Using Code Classification Techniques
An approach for the precise and comprehensive discovery of 1-day vulnerabilities

- A new way to combine improved version- and code-based approaches
  - Key techniques: code classification techniques

Design of V1SCAN

Complementing FNs

- Improved version-based approach
- Reused code classification
- Filtering out FPs
- Improved code-based approach
- Vulnerable code classification
Design of V1SCAN

Improved version-based approach

- Addressing false positives based on the reused code classification technique

Diagram:

- OSS component
  - Exactly reused functions
  - Modified functions

- Original OSS codebase
  - Exactly reused functions
  - Modified functions
  - Unused functions

Target program

- Reused
Design of V1SCAN

Improved version-based approach

- Vulnerability detection

  Third-party OSS components and versions

  e.g., Common Platform Enumeration (CPE) of NVD

  Candidate vulnerabilities

Known Affected Software Configurations

Example CPE for CVE-2014-0160
Design of V1SCAN

Improved version-based approach

• Filtering FPs (\texttt{func} : a vulnerable function of a detected vulnerability)
  \begin{itemize}
  \item \texttt{func} is \textit{exactly reused} in the target program
  \item \textit{True vulnerability} (no filtering is applied)
  \end{itemize}
Design of V1SCAN

Improved version-based approach

- Filtering FPs (func: a vulnerable function of a detected vulnerability)
  - func is not used in the target program
  - Filtering out func (i.e., false alarm)

OSS codebase

Unused functions
Design of V1SCAN

Improved version-based approach

• Filtering FPs (\texttt{func}: a vulnerable function of a detected vulnerability)
  - \texttt{func} is reused with code changes
  - Compare \texttt{func} to the vulnerable and patched functions of the vulnerability
    - \texttt{func} is more similar with the patched function → Filtering out \texttt{func} (e.g., backporting)

OSS codebase
Design of V1SCAN

Improved code-based approach

- Addressing false negatives based on the vulnerable code classification technique

Concerns of existing code-based approaches

Over 97% of the code repaired by security patches was included in one of the four locations.
Design of V1SCAN

Improved code-based approach

- Signature generation

Security patch (e.g., CVE-2019-12904)

Vulnerability signature

```
#include <libgcrypt/cipher/cipher-gcm.c>
...
+#ifdef HAVE_GCC_ATTRIBUTEAligned
+#define ATTRAligned64 __attribute__((aligned (64))
...
- static const u16 gcmR[256] = {
  - 0x0000, 0x01c2, 0x0384, 0x0246, 0x0708, 0x06ca, 0x048c,
  ...
+ static struct {
  + volatile u32 counter_head;
  ...
- for (i = 0; i < len; i += 8 * 32)
+ for (i = 0; len - i >= 8 * 32; i += 8 * 32)
```
Design of V1SCAN

Improved code-based approach

- Vulnerability detection: macro and variable
  - If the two conditions are satisfied, we conclude that 1-day vulnerabilities exist

Target program

All code lines deleted in the patch
(- code lines)

All code lines added in the patch
(+ code lines)
Design of V1SCAN

Improved code-based approach

- Vulnerability detection: function and structure
  - We first verify whether a function (or structure) similar to the vulnerable function exist
  - If the two conditions are satisfied, we conclude that 1-day vulnerabilities exist

![Diagram of Target program with similar function (structure)]

- All code lines deleted in the patch (- code lines)
- All code lines added in the patch (+ code lines)
Evaluation

Dataset

- CVE dataset (collected from NVD)
  - Vulnerable codes from 4,612 C/C++ security patches
    - Functions, structures, macros, and variables
- CPEs from all CVEs (as of August 2022)
- Target software dataset
  - Collected from GitHub
  - Popular, containing many OSS components

Table 4: Target software overview.

<table>
<thead>
<tr>
<th>IDX</th>
<th>Name</th>
<th>Version</th>
<th>#CVE</th>
<th>#OSS</th>
<th>#C/C++ Line</th>
<th>#Star</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Turicreate</td>
<td>v6.4.1</td>
<td>69</td>
<td>28</td>
<td>4,091,413</td>
<td>10.7K</td>
<td>Machine learning</td>
</tr>
<tr>
<td>S2</td>
<td>ReactOS</td>
<td>v0.4.13</td>
<td>67</td>
<td>23</td>
<td>6,419,855</td>
<td>10.8K</td>
<td>Operating system</td>
</tr>
<tr>
<td>S3</td>
<td>TizenRT</td>
<td>3.0_GBM</td>
<td>62</td>
<td>22</td>
<td>2,156,848</td>
<td>439</td>
<td>Operating system</td>
</tr>
<tr>
<td>S4</td>
<td>Aseprite</td>
<td>v1.2.25</td>
<td>53</td>
<td>12</td>
<td>846,500</td>
<td>17K</td>
<td>Animation tool</td>
</tr>
<tr>
<td>S5</td>
<td>FreeBSD</td>
<td>v12.2.0</td>
<td>30</td>
<td>47</td>
<td>14,489,534</td>
<td>6.4K</td>
<td>Operating system</td>
</tr>
<tr>
<td>S6</td>
<td>MongoDB</td>
<td>r4.2.11</td>
<td>28</td>
<td>13</td>
<td>2,822,534</td>
<td>21.5K</td>
<td>Database</td>
</tr>
<tr>
<td>S7</td>
<td>MAME</td>
<td>0228</td>
<td>24</td>
<td>26</td>
<td>4,541,014</td>
<td>5.8K</td>
<td>Emulator</td>
</tr>
<tr>
<td>S8</td>
<td>Filament</td>
<td>v1.9.9</td>
<td>16</td>
<td>16</td>
<td>1,295,918</td>
<td>13.8K</td>
<td>Rendering engine</td>
</tr>
<tr>
<td>S9</td>
<td>Godot</td>
<td>v3.2.2</td>
<td>16</td>
<td>21</td>
<td>1,298,228</td>
<td>48.1K</td>
<td>Game engine</td>
</tr>
<tr>
<td>S10</td>
<td>ArangoDB</td>
<td>v3.6.12</td>
<td>15</td>
<td>22</td>
<td>5,465,881</td>
<td>12.2K</td>
<td>Database</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>380</td>
<td>230</td>
<td>43,427,725</td>
<td>147K</td>
<td></td>
</tr>
</tbody>
</table>

\*: CVEs discovered by the version-based approach, §: Stargazers.
• Comparison targets: V0Finder [Security ’21] and MOVERY [Security ’22]
  ■ V1SCAN outperformed existing approaches
  ❖ Discovered 50% more 1-day vulnerabilities than MOVERY

<table>
<thead>
<tr>
<th>Target program</th>
<th>CVEs*</th>
<th>V1SCAN</th>
<th>MOVERY</th>
<th>V0Finder</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#TP</td>
<td>#FP</td>
<td>#FN</td>
<td>P†</td>
</tr>
<tr>
<td>Turicreate</td>
<td>36</td>
<td>32</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>ReactOS</td>
<td>29</td>
<td>26</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>23</td>
<td>19</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>MongoDB</td>
<td>14</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Filament</td>
<td>14</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TizenRT</td>
<td>10</td>
<td>9</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Aseprite</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MAME</td>
<td>8</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Godot</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ArangoDB</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>150</td>
<td>137</td>
<td>6</td>
<td>13</td>
</tr>
</tbody>
</table>

CVEs*: Total number of TPs detected by V1SCAN, MOVERY, and V0Finder,  P†: Precision,  R‡: TP detection rate.
Evaluation

Effectiveness

• Comparison targets: CENTRIS [ICSE ’21] (version-based) and VUDDY [S&P ’17] (code-based)
  ▪ V1SCAN reduced false positive ratio of the version-based approach from 77% to 4%
  ▪ V1SCAN reduced false negative ratio of the code-based approach from 49% to 9%

![Graph showing false positive and negative ratios for version-based, code-based, and V1SCAN approaches]
Conclusion

- 1-day vulnerabilities have various propagation patterns
  - E.g., propagate with code modifications or resolved after propagation

- **V1SCAN**
  - An effective approach for discovering 1-day vulnerabilities in third-party OSS components
  - V1SCAN significantly outperformed existing approaches
    - High vulnerability detection accuracy: 96% precision and 91% recall

- Equipped with vulnerability discovery results from V1SCAN
  - Developers can address threats caused by propagated vulnerabilities in OSS components