BScount: Direct Whole Patch Presence Test for Java Executables

Jiarun Dai†, Yuan Zhang†, Zheyue Jiang†, Yingtian Zhou†, Junyan Chen†, Xinyu Xing*, Xiaohan Zhang†, Xin Tan†, Min Yang†, Zhemin Yang†

†Fudan University
*Pennsylvania State University
Background

- Vulnerabilities in open-source libraries/projects could be propagated to closed-source software

Research Problem:
To check whether a given closed-source software is affected by known vulnerabilities
Patch Presence Test

• Check the presence of a patch in target software
Target Users

Government Agents

Enterprise Users

Security Companies

Developers & Testers

Third-party users of closed-source product

Security Analyst
Challenges

• 1) Need to Perform Source-to-Binary Analysis
  • Different Language layers
  • Therefore, we need to perform a hard cross-layer code equivalence test

Source code  Equivalent?  Bytecode
Challenges

- 2) Security patches may only introduce minor changes
  - Example: CVE-2018-9474, move only one line of code
  - *Thus, we need fine-grained detection methods*
Challenges

• 3) Test closed-source targets may be customized from open-source counterparts
  • Example: Android ROM, Linux Kernel
  • *Thus, patch presence test should be resilient to code customization*
**BScout**

- **BScout**: Direct Whole Patch Presence Test for *Java Executables*
  - Directly test presence of the **whole patch** in Java **Bytecode** from **Source**

**Key Idea-1**: Cross-level features to bridge the gap between source and binary.
BScout

• BScout: Direct Whole Patch Presence Test for *Java Executables*
  • Directly test presence of the whole patch in Java Bytecode from Source

**Key Idea-2:** Perform fine-grained line-level matching techniques.
BScout

- **BScout**: Direct Whole Patch Presence Test for *Java Executables*
  - Directly test presence of the whole patch in Java *Bytecode* from *Source*

**Key Idea-3**: Utilize pre-/post-patch code to neutralize customization.
Workflow

(1) Learning-based Line Segmentation

Source code Feature Lines

Post-patch source code

Target Binary

Pre-patch source code

Bytecode Feature Lines

(2) Line-level Correlative Analysis

Post-patch Line-to-Line Map

(3) Patch-derived Differential Analysis

Patch Presence Test Result

Pre-patch Line-to-Line Map

Source code Feature Lines

Patch diff
### Evaluation

#### Dataset of Android ROMs
- **150** Android Framework CVEs (2015.8 ~ 2019.7)
- **15** Android ROMs from **6** vendors
  - Google, Xiaomi, Meizu, Vivo, Oppo, Samsung

#### Dataset of Java Apps
- **15** CVEs of **11** popular 3rd-party libraries
- **261** Android Apps
- **28** Desktop/Server Apps

<table>
<thead>
<tr>
<th>Tool</th>
<th>Android ROMs</th>
<th>Java Apps</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TP  TN  FN  FP</td>
<td>Acc.</td>
<td>TP  TN  FN  FP</td>
</tr>
<tr>
<td>BScout</td>
<td>266 177 31 0</td>
<td>93.5%</td>
<td>286 410 5 0</td>
</tr>
</tbody>
</table>
Empirical Study

• Understand Real-world Patch Application Practice

2,506 Android ROMs from 7 vendors (Google, Xiaomi, Meizu, Huawei, Oppo, Vivo, Samsung)

150 Android Framework CVEs 2015.8 ~ 2019.7

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Phone Models</th>
<th>Count</th>
<th>Versions</th>
<th>Build Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>14</td>
<td>569</td>
<td>4.4.4-8.1.0</td>
<td>2014.06-2019.05</td>
</tr>
<tr>
<td>Samsung</td>
<td>24</td>
<td>468</td>
<td>5.0.0-8.1.0</td>
<td>2016.10-2018.09</td>
</tr>
<tr>
<td>Meizu</td>
<td>44</td>
<td>481</td>
<td>5.0.1-8.1.0</td>
<td>2015.06-2019.07</td>
</tr>
<tr>
<td>Xiaomi</td>
<td>45</td>
<td>464</td>
<td>4.4.4-8.1.0</td>
<td>2016.02-2019.08</td>
</tr>
<tr>
<td>Oppo</td>
<td>31</td>
<td>281</td>
<td>4.4.4-8.1.0</td>
<td>2014.11-2019.08</td>
</tr>
<tr>
<td>Vivo</td>
<td>46</td>
<td>152</td>
<td>5.0.2-8.1.0</td>
<td>2015.11-2019.05</td>
</tr>
<tr>
<td>Huawei</td>
<td>31</td>
<td>91</td>
<td>6.0.0-7.0.0</td>
<td>2016.01-2017.10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Android Version</th>
<th># of Affected CVEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android 4,*</td>
<td>40</td>
</tr>
<tr>
<td>Android 5,*</td>
<td>69</td>
</tr>
<tr>
<td>Android 6,*</td>
<td>95</td>
</tr>
<tr>
<td>Android 7,*</td>
<td>92</td>
</tr>
<tr>
<td>Android 8,*</td>
<td>50</td>
</tr>
<tr>
<td>Android 9,*</td>
<td>26</td>
</tr>
<tr>
<td>Total</td>
<td>150</td>
</tr>
</tbody>
</table>

1

Empirical Study

- **Q1**: What is the average lag for different vendors to apply a patch?

<table>
<thead>
<tr>
<th>Vendor</th>
<th># of Selected Phone Models</th>
<th># of ROMs per Model</th>
<th>Average Patch Lag per Model (day)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>12</td>
<td>20~77</td>
<td>-65~21.47</td>
</tr>
<tr>
<td>Samsung</td>
<td>16</td>
<td>10~54</td>
<td>38.16~412</td>
</tr>
<tr>
<td>Xiaomi</td>
<td>33</td>
<td>10~31</td>
<td>70.07~449.25</td>
</tr>
<tr>
<td>Meizu</td>
<td>25</td>
<td>10~29</td>
<td>85~411</td>
</tr>
<tr>
<td>Vivo</td>
<td>2</td>
<td>10~12</td>
<td>186.35~194.58</td>
</tr>
<tr>
<td>Oppo</td>
<td>10</td>
<td>11~24</td>
<td>62.71~368.89</td>
</tr>
<tr>
<td>Huawei</td>
<td>1</td>
<td>10</td>
<td>65.55</td>
</tr>
</tbody>
</table>

**Take-away:**

1. Google proactively applies patches before announcing the vulnerabilities to the public.
2. Third-party device manufactures apply patches relatively slowly.
Empirical Study

• **Q2**: Do vendors patch vulnerabilities for one model but ignore another?

<table>
<thead>
<tr>
<th>Vendor</th>
<th># of Ill-managed CVEs</th>
<th># of Ill-managed Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>24</td>
<td>12</td>
</tr>
<tr>
<td>Samsung</td>
<td>76</td>
<td>25</td>
</tr>
<tr>
<td>Meizu</td>
<td>93</td>
<td>43</td>
</tr>
<tr>
<td>Xiaomi</td>
<td>75</td>
<td>43</td>
</tr>
<tr>
<td>Oppo</td>
<td>63</td>
<td>20</td>
</tr>
<tr>
<td>Vivo</td>
<td>41</td>
<td>35</td>
</tr>
<tr>
<td>Huawei</td>
<td>33</td>
<td>32</td>
</tr>
</tbody>
</table>

**Take-away:** All vendors (including Google) have ever patched a vulnerability on one model but forgot to patch the same vulnerability on another model.

*Ill-managed Model:* models forgotten by vendors to patch some vulnerabilities

*Ill-managed CVE:* vulnerabilities forgotten by vendors to be patched
Empirical Study

• **Q3**: Do vendors correctly set security patch level?

<table>
<thead>
<tr>
<th>Vendor</th>
<th># of Negligent ROMs</th>
<th># of Diligent ROMs</th>
<th># of Prudent ROMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>112</td>
<td>182</td>
<td>185</td>
</tr>
<tr>
<td>Samsung</td>
<td>376</td>
<td>12</td>
<td>66</td>
</tr>
<tr>
<td>Meizu</td>
<td>412</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Xiaomi</td>
<td>448</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Oppo</td>
<td>173</td>
<td>19</td>
<td>24</td>
</tr>
<tr>
<td>Vivo</td>
<td>139</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Huawei</td>
<td>89</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

1) **Negligent ROMs**: some unpatched vulnerabilities at a lower patch level
2) **Diligent ROMs**: all vulnerabilities before declared patch level patched
3) **Prudent ROMs**: even patch some vulnerabilities at a higher patch level

**Take-away:**
Every vendor including Google inevitably over-claims the security patch level in some of their devices.
Empirical Study

• **Q4**: Which factors affect the application ratio for a security patch?

- Complexity of a security patch?
- Severity of a vulnerability?
- Code Customization?

**Take-away:**
It seems vulnerability severity and patch complexity are not considered by vendors in applying patches, but code customization is an obvious obstacle.
THANKS

Q&A

jrdai14@fudan.edu.cn
Evaluation

• Comparison with Function-level Similarity
  • Leverage **Centroid**\(^1\) to measure function-level similarity
  • Pre-patch and post-patch versions of 471 patch-related functions

\[
\text{When } |\alpha - \beta| > \text{threshold}, \quad \text{How to determine the threshold?}
\]

\[
\text{if } \alpha > \beta, \text{ then patch is absent, otherwise, patch is present}
\]

Detection Strategy

When the threshold increases,
1) the ratio of **can give result** drop dramatically,
2) while the ratio of **can give correct result** does not increase significantly

\(^1\) Achieving Accuracy and Scalability Simultaneously in Detecting Application Clones on Android Markets. In ICSE’14.
Evaluation

• Comparison with Version Pinning
  • Leverage LibScout\textsuperscript{[1]} and OSSPolice\textsuperscript{[2]} to do version pinning
  • Reference Set: 215 unique ROMs compiled from different tags and branches of AOSP

\begin{table}[h]
\centering
\begin{tabular}{|l|c|c|c|c|c|c|c|}
\hline
\textbf{Tool} & \textbf{Cannot Give Results} & \textbf{Can Give Results} & & & & & \\
 & \textbf{Count} & \textbf{Ratio} & \textbf{TP} & \textbf{TN} & \textbf{FP} & \textbf{FN} & \textbf{Acc.} & \textbf{FPR} \\
\hline
LibScout & 455 & 96.0\% & 12 & 1 & 0 & 6 & 68.4\% & 0\%
\hline
OSSPolice & 5 & 1.1\% & 69 & 168 & 6 & 226 & 50.5\% & 3.5\%
\hline
\end{tabular}
\end{table}

\textsuperscript{[1]} Reliable Third Party Library Detection in Android and its Security Applications. In CCS’16
\textsuperscript{[2]} Identifying Open-Source License Violation and 1-day Security Risk at Large Scale. In CCS’17
Learning-based Line Segmentation

• Split raw continuous .smali instructions into **segments**
  • **Segment: instructions generated from same Java source line**
  • Conditional Random Fields
    • Sequence Labeling
      • **S**: single instruction, **B**: begin of a segment, **M**: middle of a segment, **E**: end of a segment
Line-level Correlative Analysis

• Construct function-level line-to-line map between source code and bytecode
  • 1) Determine whether a source code line and a bytecode block are equivalent

```java
if (mActivityManager.bindBackupAgent(app, mode)) {
  if (mActivityManager.bindBackupAgent(app.packageName, mode, ... v6
if-eqz v6, :cond_b1
```

```java
12-  if (mActivityManager.bindBackupAgent(app, mode)) {
13+  if (mActivityManager.bindBackupAgent(app.packageName, mode, UserHandle.USER_OWNER)) {
```

```java
iget-object v6, p0, L..ActivityManagerService;->mActivityManager:L..IActivityManager;
invoke-interface {v6, p1, p2}, L..IActivityManager;->bindBackupAgent(L..ApplicationInfo;I)Z
move-result v6
if-eqz v6, :cond_b1
```
Line-level Correlative Analysis

• Construct function-level line-to-line map between source code and bytecode
  • 2) Map all source code lines and instruction blocks in the scope of the whole method
Patch-derived Differential Analysis

• Use following presence test strategies for a patch