MVP: Detecting Vulnerabilities using Patch-Enhanced Vulnerability Signatures

Yang Xiao¹,², Bihuan Chen³, Chendong Yu¹,², Zhengzi Xu⁴, Zimu Yuan¹,², Feng Li¹,², Binghong Liu¹,², Yang Liu⁴, Wei Huo¹,², Wei Zou¹,², Wenchang Shi⁵

1. Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
2. School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
3. School of Computer Science and Shanghai Key Laboratory of Data Science, Fudan University, China
4. School of Computer Science and Engineering, Nanyang Technological University, Singapore
5. Renmin University of China, Beijing, China
Background

• Vulnerabilities can be exploited to attack software systems, threatening system security.
  • Detect and patch vulnerabilities as early as possible.

• Reusing code base or sharing code logic is common.
  • E.g., Same action for processing different kinds of files (bmp/dib/...) in ImageMagick.

• Recurring vulnerabilities (share the similar characteristics with each other) widely exist but remain undetected.
Existing Approaches

• **Clone-based approaches**
  • They consider the recurring vulnerability detection problem as a code clone detection problem

• **Function matching based approaches**
  • They use vulnerable functions in a known vulnerability as the signature and detect code clones to those vulnerable functions
  • [16 ICSE] SourcererCC: Scaling Code Clone Detection to Big-Code
  • [18 ICSE] CCAligner: A Token Based Large-Gap Clone Detector
patch for CVE-2017-14041

@@ -1185,7 +1185,7 @@
 opj_image_t *
 pgxtoimage(const char *filename, opj_cparameters_t *parameters)
 }

-void pgxtoimage(const char *filename, opj_cparameters_t *parameters) {
+void pgxtoimage(const char *filename, opj_cparameters_t *parameters) {
      FILE *f = NULL;
      ...  
      fseek(f, 0, SEEK_SET);
-     if (fscanf(f, "PG\%[ \t]\%c%[ \t+]%d%[ \t]d%[ \t]d%", temp, &endian1,
+     if (fscanf(f, "PG%31\%[ \t]\%c%31[ \t+]%d%31%[ \t]d%31%[ \t]d%", temp, &endian1,
                  &endian2, signtmp, &prec, temp, &w, temp, &h) != 9) {
           fclose(f);
           fprintf(stderr,

//vulnerable function: pgxtoimage (src/bin/jp2/convert.c)
1 opj_image_t* pgxtoimage(const char *filename, opj_cparameters_t *parameters) {
2 {
3     FILE *f = NULL;
4     ...  
5     fseek(f, 0, SEEK_SET);
6     if (fscanf(f, "PG\%[ \t]\%c%[ \t+]%d%[ \t]d%[ \t]d%", temp, &endian1,
7                  &endian2, signtmp, &prec, temp, &w, temp, &h) != 9) {
8         fclose(f);
9         fprintf(stderr,
10            "ERROR: Failed to read the right number of element from the fscanf() function!\n"");
11         return NULL;
12     }

//target function (found by MVP): pgxtoimage (src/bin/jpwl/convert.c)
1 opj_image_t* pgxtoimage(const char *filename, opj_cparameters_t *parameters) {
2 {
3     FILE *f = NULL;
4     ...  
5     fseek(f, 0, SEEK_SET);
6     if (fscanf(f, "PG%31\%[ \t]\%c%31[ \t+]%d%31%[ \t]d%31%[ \t]d%", temp, &endian1,
7                  &endian2, signtmp, &prec, temp, &w, temp, &h) != 9) {
8         fprintf(stderr,
9            "ERROR: Failed to read the right number of element from the fscanf() function!\n"");
10        fclose(f);
11        return NULL;
12    }

ReDeBug
Line 5 – line 8 => hash r1  X
Line 6 – line 9 => hash r2  X
Line 7 – line 10 => hash r3 X

VUDDY
All statements => hash v  X
When Sim(V,P) is large, existing approaches can introduce high false positives. Sim(V,P) is above 70% for 91.3% of pairs.

When Sim(V,T) is small, existing approaches may introduce high false negatives. 35.1% of pairs <V, T> have a Sim(V,T) of lower than 70% and existing approaches miss most of them.

Note: Sim(f1, f2) denotes the similarity score between function f1 and f2.
Challenges

• C1: How to distinguish already patched vulnerabilities to reduce false positives.

• C2: How to precisely generate the signature of a known vulnerability to reduce both false positives and false negatives.
Challenges

C1: How to distinguish already patched vulnerabilities to reduce false positives.

C2: How to precisely generate the signature of a known vulnerability to reduce both false positives and false negatives.

Approach

- Vulnerability signature + patch signature
- Novel slicing method + entropy-based statement selection
- Syntactic + semantic
- Abstraction + normalization
Overview of MVP

1. Extracting Function Signature
   - Parsing and Analyzing Function
     - Target System
     - Abstract Syntax Tree
     - Program Dependence Graph
   - Abstracting and Normalizing Function
     - Abstracted and Normalized Function
   - Generating Function Signature
     - Function Signature
   - 3. Detecting Vulnerability
     - Potentially Vulnerable Function

2. Extracting Vulnerability and Patch Signatures
   - Identifying Code Changes
     - Security Patch
     - Deleted and Added Statements
     - Vulnerable and Patched Functions
   - Computing Slices to Changed Code
   - Semantically Related Statements
   - Generating Vulnerability and Patch Signatures
     - Vulnerability and Patch Signatures
Formal parameters -> PARAM
Local variables -> VARIABLES
String -> STRING (except format string)
Target information:
- Changed files and its corresponding commits
  - wma.c, 0cb2ab8bd (vul ver), cac414969 (pat ver)
- Vulnerable functions, patched functions
  - Changed function: WDA_TxPacket
- Deleted/Added statements
  - Line 18 – 22 (add lines)
Too many statements are included while some of them are not relevant to the vulnerability.
Backward slicing
• Perform normal backward slicing on PDG

Forward slicing
• Assignment statement
  • Normal forward slicing
• Conditional statement
  • Conduct backward slicing on data dependencies in the PDG to obtain the direct source for each variable/parameter
  • Set each statement in the first step as the slicing criterion, and perform forward slicing on data dependencies
  • Only if the previous forward slicing result is empty, perform normal forward slicing on control dependencies.
• Return statement
  • No need for forward slicing
• Others
  • Similar to conditional statement, following the same first and second steps for conditional statements.
The number of statements in $V_{syn}$ varies for different patches. If the number of statements is very large, $V_{syn}$ may introduce noise and result in false negatives.

If $T > t_{\text{max}}$, we iteratively remove from $V_{syn}$ statements which are farthest from the slicing criterion on the PDG until $T$ is less than $t_{\text{max}}$. 

\[ V_{syn} = S_{del} \cup (S_{vul} \cap S_{add}) \]  
\[ V_{sem} = \{(s_1, s_2, \text{type}) | s_1, s_2 \in V_{syn}\} \]  
\[ T_{sem} = \{(s_1, s_2, \text{type}) | s_1, s_2 \in S_{add}\} \]  
\[ P_{syn} = S_{add} \setminus S_{vul} \]  
\[ P_{sem} = T_{sem} \setminus F_{vul} \]  
\[ F_{vul} = \{(s_1, s_2, \text{type}) | s_1, s_2 \in S_{vul}\} \]
• C1. The target function must contain all deleted statements, if any; i.e., $\forall h \in S_{del}, h \in f_{syn}$.
• C2. The signature of the target function matches the vulnerability signature at the syntactic level; i.e., $\frac{|V_{syn} \cap f_{syn}|}{|V_{syn}|} > t_1$.
• C3. The signature of the target function does not match the patch signature at the syntactic level; i.e., $\frac{|P_{syn} \cap f_{syn}|}{|P_{syn}|} \leq t_2$.
• C4. The signature of the target function matches the vulnerability signature at the semantic level; i.e., $\frac{|V_{sem} \cap f_{sem}|}{|V_{sem}|} > t_3$.
• C5. The signature of the target function does not match the patch signature at the semantic level; i.e., $\frac{|P_{sem} \cap f_{sem}|}{|P_{sem}|} \leq t_4$. 
## Dataset

<table>
<thead>
<tr>
<th>Target System</th>
<th>Version</th>
<th>Line (#)</th>
<th>Function (#)</th>
<th>Domain</th>
<th>NVD (#)</th>
<th>Commit (#)</th>
<th>Total (#)</th>
<th>Changed Function (#)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux kernel</td>
<td>v4.18</td>
<td>18,298,218</td>
<td>435,734</td>
<td>Operating System Kernel</td>
<td>1,628</td>
<td>17,618</td>
<td>18,495</td>
<td>19,904</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>12.0</td>
<td>7,460,955</td>
<td>140,163</td>
<td>Operation System Kernel</td>
<td>160</td>
<td>3,656</td>
<td>3,716</td>
<td>7,703</td>
</tr>
<tr>
<td>ImageMagick</td>
<td>7.0.8-27</td>
<td>461,843</td>
<td>4,229</td>
<td>Image Processing</td>
<td>79</td>
<td>628</td>
<td>704</td>
<td>915</td>
</tr>
<tr>
<td>OpenJPEG</td>
<td>2.3.0</td>
<td>245,113</td>
<td>4,390</td>
<td>Image Processing</td>
<td>17</td>
<td>137</td>
<td>142</td>
<td>309</td>
</tr>
<tr>
<td>LibTIFF</td>
<td>v4.0-9</td>
<td>82,985</td>
<td>1,413</td>
<td>Image Processing</td>
<td>46</td>
<td>175</td>
<td>193</td>
<td>343</td>
</tr>
<tr>
<td>Libarchive</td>
<td>v3.3.3</td>
<td>194,050</td>
<td>3,283</td>
<td>Compression</td>
<td>15</td>
<td>141</td>
<td>152</td>
<td>353</td>
</tr>
<tr>
<td>Libming</td>
<td>0.4.8</td>
<td>73,888</td>
<td>2,375</td>
<td>Flash Processing</td>
<td>17</td>
<td>39</td>
<td>53</td>
<td>147</td>
</tr>
<tr>
<td>Libav</td>
<td>12.3</td>
<td>607,326</td>
<td>11,277</td>
<td>Video Processing</td>
<td>80</td>
<td>763</td>
<td>813</td>
<td>1,467</td>
</tr>
<tr>
<td>Asterisk</td>
<td>16.6.0</td>
<td>995,874</td>
<td>19,202</td>
<td>Communication Toolkit</td>
<td>7</td>
<td>556</td>
<td>533</td>
<td>2,080</td>
</tr>
<tr>
<td>Qcacld-2.0</td>
<td>le.4.0.4</td>
<td>490,638</td>
<td>7,541</td>
<td>WLAN Driver</td>
<td>44</td>
<td>561</td>
<td>576</td>
<td>1,157</td>
</tr>
</tbody>
</table>

| Total          | –       | 28,910,890| 629,607      | –                             | 2,093   | 24,274     | 25,377    | 34,378               |
## Result

<table>
<thead>
<tr>
<th>Target System</th>
<th>GT (##)</th>
<th>ReDeBug</th>
<th></th>
<th>VUDDY</th>
<th></th>
<th>MVP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TP</td>
<td>FP</td>
<td>FN</td>
<td>Precision</td>
<td>Recall</td>
<td>TP</td>
</tr>
<tr>
<td>Linux kernel</td>
<td>32</td>
<td>12</td>
<td>286</td>
<td>20</td>
<td>4.0%</td>
<td>37.5%</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>11</td>
<td>7</td>
<td>86</td>
<td>4</td>
<td>7.5%</td>
<td>63.6%</td>
</tr>
<tr>
<td>ImageMagick</td>
<td>16</td>
<td>7</td>
<td>14</td>
<td>9</td>
<td>33.3%</td>
<td>43.7%</td>
</tr>
<tr>
<td>OpenJPEG</td>
<td>16</td>
<td>10</td>
<td>7</td>
<td>6</td>
<td>58.8%</td>
<td>62.5%</td>
</tr>
<tr>
<td>LibTIFF</td>
<td>8</td>
<td>6</td>
<td>11</td>
<td>2</td>
<td>35.3%</td>
<td>75.0%</td>
</tr>
<tr>
<td>Libarchive</td>
<td>5</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>14.3%</td>
<td>20.0%</td>
</tr>
<tr>
<td>Libming</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Libav</td>
<td>6</td>
<td>2</td>
<td>10</td>
<td>4</td>
<td>16.7%</td>
<td>33.3%</td>
</tr>
<tr>
<td>Asterisk</td>
<td>7</td>
<td>4</td>
<td>30</td>
<td>3</td>
<td>11.8%</td>
<td>57.1%</td>
</tr>
<tr>
<td>Qcacl-2.0</td>
<td>7</td>
<td>1</td>
<td>44</td>
<td>6</td>
<td>2.2%</td>
<td>14.3%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>111</td>
<td>50</td>
<td>499</td>
<td>61</td>
<td>9.1%</td>
<td>45.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target System</th>
<th></th>
<th>ReDeBug</th>
<th></th>
<th>VUDDY</th>
<th></th>
<th>MVP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux kernel</td>
<td>1,883 s</td>
<td>0.68 ms</td>
<td>0.01 ms</td>
<td>6,974 s</td>
<td>3,846.17 ms</td>
<td>83.10 ms</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>1,008 s</td>
<td>0.94 ms</td>
<td>0.03 ms</td>
<td>6,868 s</td>
<td>4,966.36 ms</td>
<td>63.24 ms</td>
</tr>
<tr>
<td>ImageMagick</td>
<td>35 s</td>
<td>1.27 ms</td>
<td>0.01 ms</td>
<td>221 s</td>
<td>7,228.69 ms</td>
<td>8.52 ms</td>
</tr>
<tr>
<td>OpenJPEG</td>
<td>11 s</td>
<td>1.40 ms</td>
<td>0.01 ms</td>
<td>251 s</td>
<td>5,697.18 ms</td>
<td>84.51 ms</td>
</tr>
<tr>
<td>LibTIFF</td>
<td>7 s</td>
<td>3.62 ms</td>
<td>0.01 ms</td>
<td>53 s</td>
<td>6,036.26 ms</td>
<td>108.81 ms</td>
</tr>
<tr>
<td>Libarchive</td>
<td>20 s</td>
<td>1.31 ms</td>
<td>0.01 ms</td>
<td>121 s</td>
<td>5,263.15 ms</td>
<td>39.47 ms</td>
</tr>
<tr>
<td>Libming</td>
<td>9 s</td>
<td>3.77 ms</td>
<td>0.01 ms</td>
<td>47 s</td>
<td>3,981.13 ms</td>
<td>113.21 ms</td>
</tr>
<tr>
<td>Libav</td>
<td>41.4 s</td>
<td>1.11 ms</td>
<td>0.01 ms</td>
<td>206 s</td>
<td>3,569.50 ms</td>
<td>29.52 ms</td>
</tr>
<tr>
<td>Asterisk</td>
<td>45.5 s</td>
<td>3.94 ms</td>
<td>0.01 ms</td>
<td>156 s</td>
<td>7,335.83 ms</td>
<td>125.70 ms</td>
</tr>
<tr>
<td>Qcacl-2.0</td>
<td>26 s</td>
<td>1.04 ms</td>
<td>0.01 ms</td>
<td>57 s</td>
<td>5,499.53 ms</td>
<td>517.36 ms</td>
</tr>
</tbody>
</table>
## Result

<table>
<thead>
<tr>
<th>Approach</th>
<th>10%*</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
<th>60%</th>
<th>70%</th>
<th>80%</th>
<th>90%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVP</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>5</td>
<td>10</td>
<td>14</td>
<td>13</td>
<td>26</td>
<td>8</td>
</tr>
<tr>
<td>ReDeBug</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>11</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>VUDDY</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>SourcererCC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>18</td>
<td>30</td>
<td>8</td>
</tr>
<tr>
<td>CCAignner</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>14</td>
<td>29</td>
<td>6</td>
</tr>
<tr>
<td>VulDeePecker</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Devign</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>Coverity</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Checkmarx</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ground Truth</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>10</td>
<td>16</td>
<td>18</td>
<td>30</td>
<td>8</td>
</tr>
</tbody>
</table>

* x% denotes the similarity score between vulnerable function and its corresponding matched target function.
Thank you!

- Contact: xiaoyang@iie.ac.cn