Diagnosis and Emergency Patch Generation for Integer Overflow Exploits

Tielei Wang, Chengyu Song, Wenke Lee
Georgia Institute of Technology

Motivation

• A common task for defenders is to analyze the captured exploits and generate patches, e.g.,
Scope of Work

• Integer Overflow Vulnerabilities

Challenge of Diagnosis

• Benign integer overflows are very common in binaries
Challenge of Diagnosis

• Benign integer overflows are very common in binaries
  – Random number generation, hash function, compiler optimization

```c
int rand(void) {
    next = next * 1103515245 + 12345;
    return (unsigned int)(next/65536) % 32768;
}
```

A simple random number generation function deliberately makes use of integer overflows

Challenge of Patch Generation

- Signature/Filter e.g., len of Input < 1024
- struct.file !=0
- Input Entry Point e.g., fread()
- Vulnerability Site
Challenge of Patch Generation

- Highly complicated input formats make it very hard to identify malicious inputs at the entry point

Our Goals

- Exploit Diagnosis
  – Given an exploit instance, diagnose whether it is against integer overflow vulnerabilities
- Patch Generation
  – Given the integer overflow vulnerability, generate patches to temporarily protect the vulnerable program
Exploit Diagnosis

- Intuition:
  - use dynamic taint analysis to track the propagation of an overflowed value
  - identify harmful integer overflows according to how the program uses them

CVE-2012-2026 (Adobe Flash Professional CS)
Exploit Diagnosis

CVE-2012-2026 (Adobe Flash Professional CS)

Another example

CVE-2012-1149 (OpenOffice.Org)
Patch Generation

• Intuition:
  – RemEDIATE vulnerabilities by reusing existing error handlers inside the program, rather than filtering the malicious inputs at the entry point

Local Error Virtualization

Example For Patch Generation

HGLOBAL WinSalBitmap::ImplCreateDIB( const Size& rSize, USHORT nBits, const BitmapPalette& rPal )
{
...
const ULONG nImageSize = AlignedWidth4Bytes( nBits * rSize.Width() ) * rSize.Height();
const USHORT nColors = ( nBits <= 8 ) ? ( 1 << nBits ) : 0;

hDIB = GlobalAlloc( GHND, sizeof(BITMAPINFOHEADER) + nColors * sizeof(RGBQUAD) + nImageSize );
...

CVE-2012-1149 (OpenOffice.Org)
Example For Patch Generation

• There is an existing “validation” check in the program

HGLOBAL WinSalBitmap::ImplCreateDIB( const Size& rSize, USHORT nBits, const BitmapPalette& rPal )
{
  ..
  HGLOBAL hDIB = 0;
  if ( rSize.Width() && rSize.Height() )
  {
    const ULONG nImageSize = AlignedWidth4Bytes( nBits * rSize.Width() ) * rSize.Height();
    const USHORT nColors = ( nBits <= 8 ) ? 1 << nBits : 0;
    hDIB = GlobalAlloc( GHND, sizeof( BITMAPINFOHEADER ) + nColors * sizeof( RGBQUAD ) + nImageSize );
  }
  return hDIB;
}

CVE-2012-1149 (OpenOffice.Org)

Example For Patch Generation

• Deploy patches at existing check points, and reuse the existing error handling code to survive the attack

HGLOBAL WinSalBitmap::ImplCreateDIB( const Size& rSize, USHORT nBits, const BitmapPalette& rPal )
{
  ..
  HGLOBAL hDIB = 0;
  if ( rSize.Width() && rSize.Height() && CurrentExecutionContextWillNotTriggerTheIntegerOverflow() )
  {
    const ULONG nImageSize = AlignedWidth4Bytes( nBits * rSize.Width() ) * rSize.Height();
    const USHORT nColors = ( nBits <= 8 ) ? 1 << nBits : 0;
    hDIB = GlobalAlloc( GHND, sizeof( BITMAPINFOHEADER ) + nColors * sizeof( RGBQUAD ) + nImageSize );
  }
  return hDIB;
}

CVE-2012-1149 (OpenOffice.Org)
Pattern I

Existing “Validations” Before Integer Overflows

```c
if (x < 0)
    { return -1;
    }
    ....
size = x * y * 4 + 4;
```

Real World Integer Overflows

```c
if (Bitmap_Head.biWidth < 0)
    { g_set_error (error, G_FILE_ERROR,G_FILE_ERROR_FAILED,
      _("'%s' is not a valid BMP file"),
      gimp_filename_to_utf8 (filename));
    return -1;
    }
    ....
rowbytes= ((Bitmap_Head.biWidth * Bitmap_Head.biBitCnt - 1) / 32) * 4 + 4;
    ....
buffer = g_malloc (rowbytes);
```

CVE-2009-1570 (GIMP)
if (p_sys->i_track_id < 0) { 
    input_item_node_AppendNode( p_input_node, p_new_node );
    vlc_gc_decref( p_new_input );
    return true;
}

input_item_t **pp;
pp = realloc( p_sys->pp_tracklist, (p_sys->i_track_id + 1) * sizeof(*pp) );

CVE-2011-2194 (VLC Player)

......
wpattern_malloc = wpattern = (wchar_t *) malloc ((n + 1) * sizeof(wchar_t));

CVE-2011-1071 (Glibc)
Real World Integer Overflows

```c
if (args->buffer_count < 1) {
    DRM_ERROR("execbuf2 with %d buffers\n", args->buffer_count);
    return -EINVAL;
}
exec2_list = kmalloc(sizeof(*exec2_list)*args->buffer_count,
                     GFP_KERNEL | __GFP_NOWARN | __GFP_NORETRY);
```

CVE-2013-0913 (Linux Kernel)

Remediation Policy I

- Enhance existing checks before integer overflows

```c
if (x< 0) {
    return -1;
}
if (x< 0 || IntegerOverflowToHappen()) {
    return -1;
}
.....
size = x * y* 4 + 4;
size = x* y* 4 + 4;
```

Vulnerable Code

Remediated Code
Remediation Policy I

• Utilize backwards-slicing to identify existing checks before the integer overflows

• Use symbolic execution to compute the weakest preconditions
  – Check whether a concrete execution context satisfies the vulnerability trigger condition
  – If so, force the program to run the error handler

Pattern II
Existing “Validations” After Integer Overflows

```c
size = x * y * 4 + 4;
if (size < 0)
{
    return -1;
}
....
```
Real World Integer Overflows

```
space = dp->tdir_count * datawidth[dp->tdir_type];
if (space <= 0) {
    printf("\n");
    Error("Invalid count for tag %u", dp->tdir_tag);
}
```

CVE-2010-4665 (Libtiff)

---

```
bitmap_size = glyph->bpr * glyph->bbox.height;
if (bitmap_size > 0xFFFFU )
{
    FT_ERROR(( "bdf_parse_glyphs: " ERRMSG4, lineno ));
    error = BDF_Err_Bbx_Too_Big;
    goto Exit;
}
```

CVE-2012-1144 (Freetype2)
Remediation Policy II

• Reuse existing checks after integer overflow

```c
size = x * y * 4 + 4;
if (size < 0)
{
    return -1;
}
....
```

```c
size = x * y * 4 + 4;
SetAlarmIfOverflowed();
if (size < 0) || AlarmIfTrue()
{
    return -1;
}
....
```

Vulnerable Code
Remediated Code

Remediation Policy II

• Utilize forwards-slicing to identify existing checks after the integer overflows

• Monitor the integer overflow operation, and set an alarm variable if it really overflows at runtime

• Check the alarm variable at the validation check point. If the alarm is True, alter the program control flow to error handler
Pattern III
Existing “Validations” On Allocation Results

```
size = x * y * 4 + 4;
p = malloc(size);
if(p==NULL)
  goto error;
```

Real World Integer Overflows

```
PRUint32 destlen = ((srclen + 2)/3) * 4;
dest = (char *)PR_MALLOC(destlen + 1);
if((char *)0 == dest )
{
  return (char *)0;
}
```

CVE-2009-2463 (Firefox)
Real World Integer Overflows

```c
length=(size_t) matte_image->bytes_per_line*matte_image->height*matte_image->depth;
matte_image->data=(char *) malloc(length);
if (matte_image->data == (char *) NULL)
```

CVE-2009-1882 (ImageMagic)

---

Real World Integer Overflows

```c
skb = sock_wmalloc(sk, NET_SKB_PAD + sizeof(struct iphdr) + uhlen + session->hdr_len + sizeof(ppph) + total_len, 0, GFP_KERNEL);
if (!skb)
    goto error_put_sess_tun;
```

CVE-2010-4160 (Linux Kernel)
Real World Integer Overflows

coef = (MYFLT *)malloc((hdr.npoles+hdr.nvals)*sizeof(MYFLT));
if (coef==NULL) {
    printf("memory allocation failure\n");
    exit(1);
}

CVE-2012-2106 (CSound)

Remediation Policy III

- Reuse the memory allocation failure error code

<table>
<thead>
<tr>
<th>Vulnerable Code</th>
<th>Remediated Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>size = x * y * 4 + 4; p = malloc(size); if(p==NULL) goto error;</td>
<td>size = x * y * 4 + 4; p = malloc(size); if(p==NULL) goto error; SetAlarmIfOverflowed(); if(AlarmIsTrue()) p = NULL else p = malloc(size);</td>
</tr>
</tbody>
</table>
Remediation Policy III

• Utilize forwards-slicing to determine whether the integer overflow will affect memory allocations

• Monitor the integer overflow operation, and set an alarm variable if it really overflows at runtime

• Check the alarm variable before memory allocation. If the alarm is True, force the program to run the branch for memory allocation failure

System Overview
Evaluation

<table>
<thead>
<tr>
<th>Software</th>
<th>Description</th>
<th>Version</th>
<th>Open Source</th>
<th>CVE ID</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Openoffice.org</td>
<td>Office productivity software suite</td>
<td>3.3.20</td>
<td>Y</td>
<td>CVE-2012-1149</td>
<td>ODT</td>
</tr>
<tr>
<td>VLC</td>
<td>Multimedia player</td>
<td>1.1.0</td>
<td>Y</td>
<td>CVE-2011-2194</td>
<td>XSPF</td>
</tr>
<tr>
<td>Yahoo Messenger</td>
<td>Instant messaging</td>
<td>115.0.152</td>
<td>N</td>
<td>CVE-2012-0268</td>
<td>JPEG</td>
</tr>
<tr>
<td>ACDSee</td>
<td>Image viewer</td>
<td>14.1</td>
<td>N</td>
<td>CVE-2012-1197</td>
<td>BMP</td>
</tr>
<tr>
<td>Opera</td>
<td>Web browser</td>
<td>11.6</td>
<td>N</td>
<td>CVE-2012-1003</td>
<td>HTML</td>
</tr>
<tr>
<td>Adobe Flash Player</td>
<td>Web browser plug-in</td>
<td>10.0.42.34</td>
<td>N</td>
<td>CVE-2010-2170</td>
<td>SWF</td>
</tr>
<tr>
<td>Adobe Reader</td>
<td>PDF viewer</td>
<td>9.1.3</td>
<td>N</td>
<td>CVE-2009-3549</td>
<td>PDF</td>
</tr>
<tr>
<td>RealPlayer</td>
<td>Multimedia player</td>
<td>SP 1.1</td>
<td>N</td>
<td>CVE-2010-3000</td>
<td>FLV</td>
</tr>
<tr>
<td>QuickTime Player</td>
<td>Multimedia Player</td>
<td>7.1.3</td>
<td>N</td>
<td>CVE-2007-0114</td>
<td>MPEG-4</td>
</tr>
<tr>
<td>Microsoft Liner</td>
<td>Key component of Microsoft Visual Studio</td>
<td>10.00.3039303</td>
<td>N</td>
<td>N/A</td>
<td>PE</td>
</tr>
</tbody>
</table>

Summary: 10 integer overflows and 10 different input formats.

Use 10 programs on Windows to evaluate our system. We obtained the exploit samples from different sources.

Attack Diagnosis Results

<table>
<thead>
<tr>
<th>Software</th>
<th>Integer Overflow Vulnerability</th>
<th>Module</th>
<th>Offset</th>
<th># Overflow Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>Openoffice.org</td>
<td>imul edi, edx</td>
<td>vclm.dll</td>
<td>Ox1ad49f</td>
<td>1122</td>
</tr>
<tr>
<td>VLC</td>
<td>lea esi, ptr [ecx+4+0x4]</td>
<td>(playlist_plugin.dll)</td>
<td>Oxac09</td>
<td>423</td>
</tr>
<tr>
<td>Yahoo Messenger</td>
<td>imul eax, ebx</td>
<td>Y(image.dll)</td>
<td>Ox21551</td>
<td>354</td>
</tr>
<tr>
<td>ACDSee</td>
<td>imul ebp, ecx</td>
<td>IDE.ACDISd.dll</td>
<td>Ox59639</td>
<td>288</td>
</tr>
<tr>
<td>Opera</td>
<td>imul eax, dword ptr [esp+0xc]</td>
<td>opera.dll</td>
<td>Ox88892b</td>
<td>428</td>
</tr>
<tr>
<td>Adobe Flash Player</td>
<td>imul eax, ecx</td>
<td>Flash10.exe</td>
<td>Ox9165e</td>
<td>860</td>
</tr>
<tr>
<td>Adobe Reader</td>
<td>lea edx, ptr [ecx+4+0x48]</td>
<td>Acrobat32.dll</td>
<td>Oxax64x</td>
<td>1682</td>
</tr>
<tr>
<td>RealPlayer</td>
<td>imul ecx, ecx, 0x23</td>
<td>tfw.dll</td>
<td>Ox88e4</td>
<td>381</td>
</tr>
<tr>
<td>QuickTime Player</td>
<td>add ecx, edi</td>
<td>QuickTime.qts</td>
<td>Ox295a74</td>
<td>567</td>
</tr>
<tr>
<td>Microsoft Liner</td>
<td>lea edi, ptr [eax+eax+8]</td>
<td>linker.exe</td>
<td>Oxax2c10</td>
<td>86</td>
</tr>
</tbody>
</table>

Accurately locate the harmful integer overflows
Patch Generation Results

Successfully generated patches for the ten integer overflows using different policies

<table>
<thead>
<tr>
<th>Software</th>
<th>Relevant Checks</th>
<th>Validation Checks</th>
<th># Final Patches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Openoffice.org</td>
<td>17</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>Yahoo Messenger</td>
<td>14</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>ACDSsee</td>
<td>10</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>Opera</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>VLC</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Adobe Reader</td>
<td>23</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Microsoft Linker</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Summary: successfully fixed these 7 vulnerabilities by using Policy I.

Performance

Generate patches in minutes. And Patches introduce trivial performance overhead

<table>
<thead>
<tr>
<th>Software</th>
<th>Diagnosis(s)</th>
<th>Tracing(s)</th>
<th>Slicing(s)</th>
<th>Patching(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yahoo Messenger</td>
<td>57</td>
<td>164</td>
<td>16</td>
<td>6.3</td>
</tr>
<tr>
<td>OpenOffice.org</td>
<td>181</td>
<td>210</td>
<td>53</td>
<td>10.2</td>
</tr>
<tr>
<td>ACDSsee</td>
<td>123</td>
<td>206</td>
<td>18</td>
<td>8.8</td>
</tr>
<tr>
<td>Opera</td>
<td>105</td>
<td>332</td>
<td>49</td>
<td>6.5</td>
</tr>
<tr>
<td>VLC</td>
<td>112</td>
<td>134</td>
<td>28</td>
<td>8.6</td>
</tr>
<tr>
<td>Adobe Reader</td>
<td>99</td>
<td>361</td>
<td>71</td>
<td>21.5</td>
</tr>
<tr>
<td>Adobe Flash Player</td>
<td>144</td>
<td>344</td>
<td>52</td>
<td>N/A</td>
</tr>
<tr>
<td>QuickTime</td>
<td>78</td>
<td>217</td>
<td>73</td>
<td>N/A</td>
</tr>
<tr>
<td>RealPlayer</td>
<td>93</td>
<td>228</td>
<td>51</td>
<td>N/A</td>
</tr>
<tr>
<td>Microsoft Linker</td>
<td>37</td>
<td>66</td>
<td>27</td>
<td>12.1</td>
</tr>
</tbody>
</table>

Summary: diagnosing and patching were completed in minutes.

<table>
<thead>
<tr>
<th>Software</th>
<th>Normal (µs)</th>
<th>Malicious (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yahoo Messenger</td>
<td>3190</td>
<td>4503</td>
</tr>
<tr>
<td>OpenOffice.org</td>
<td>5028</td>
<td>6572</td>
</tr>
<tr>
<td>ACDSsee</td>
<td>1241</td>
<td>2442</td>
</tr>
<tr>
<td>Opera</td>
<td>727</td>
<td>761</td>
</tr>
<tr>
<td>VLC</td>
<td>597</td>
<td>1524</td>
</tr>
<tr>
<td>Adobe Reader</td>
<td>306</td>
<td>509</td>
</tr>
<tr>
<td>MS linker</td>
<td>1660</td>
<td>1819</td>
</tr>
</tbody>
</table>
Conclusion

• Propose a taint-based approach to pinpoint integer overflow vulnerabilities from an exploit instance
• Propose local error virtualization to generate emergency patches for integer overflow vulnerabilities
• Evaluation with 10 real world programs shows the effectiveness and the efficiency of our prototype system

Limitations

• Completeness and Soundness
  – Detection
  – Patch
• It is a heuristic-based system, and it’s purpose is to provide a temporary protection when there is no official patch available