Towards Efficient Heap Overflow Discovery

Xiangkun Jia, Chao Zhang, Purui Su,
Yi Yang, Huafeng Huang, Dengguo Feng

TCA/SKLCS lab, Institute of Software, Chinese Academy of Sciences,
University of Chinese Academy of Sciences,
Institute for Network Science and Cyberspace, Tsinghua University
Introduction

Motivation

• Stack overflow exploits are rare
  – Defenses are deployed, e.g., ASLR, shadow stack, Stackguard, StackArmor, etc.

• Heap overflows vulnerabilities become popular and realistic
  – Attack techniques are developing, e.g., dword shoot, heap spray, heap fengshui, etc.
Introduction

Existing methods

• Runtime detections are passive
  - rely on triggering vuls

• Fuzzers are sometimes blind
  - just bypass vuls because of value

• Dynamic analysis is lack of a complete model
  - find heap overflow by integer overflow

X = input()
buffer1 = malloc(X+2)
//Heap overflow only if X+2 is integer overflow but X+1 not
memcpy(buffer1, “xxxx”, X+1)
Introduction

Goal

• More proactive
  – Find vulnerabilities before attackers

• More targeted
  – Focus on heap overflow vulnerabilities

• More complete
  – Build a complete model of vulnerabilities
Background

Root cause: Spatial inconsistency

- Example of a heap overflow
  - Heap allocation operations (obj, size_{allocation})
  - Heap access operations (ptr, offset_{ptr}, size_{access})
- Heap overflow happens
  - When input size = SIZE

Heap overflow condition: Range_{access} > Range_{obj}
Observation: Controllable attributes

- Types of heap overflow (based on controllable attributes)

<table>
<thead>
<tr>
<th></th>
<th>Alloc controllable</th>
<th>Access controllable</th>
<th>How to overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>N</td>
<td>N</td>
<td>--/inherent errors</td>
</tr>
<tr>
<td>S2</td>
<td>N</td>
<td>Y</td>
<td>Copy more data/Copy to outside offset</td>
</tr>
<tr>
<td>S3</td>
<td>Y</td>
<td>N</td>
<td>Allocate small buffer/allocation size is integer overflow</td>
</tr>
<tr>
<td>S4</td>
<td>Y (bytes_1)</td>
<td>Y (bytes_2)</td>
<td>Change the value of two sides independently</td>
</tr>
<tr>
<td>S5</td>
<td>Y (bytes_0)</td>
<td>Y (bytes_0)</td>
<td>Check IO2BO (e.g., (x+2, x+1))</td>
</tr>
</tbody>
</table>
Overview of HOTracer

1. Get execution traces
2. Identify heap operations
3.1 Track spatial attributes
3.2 Track controllable (taint) attributes
4. Build vulnerability conditions
5. Generate PoC inputs
Design

Challenges

1. Get execution traces
2. Identify heap operations
3.1 Track spatial attributes
3.2 Track controllable (taint) attributes
4. Build vulnerability conditions
5. Generate PoC inputs

Make it efficient!

1. Get representative traces efficiently
2. Handle custom function implementations
3. Pair heap operations accurately
4. Use fine-grained taint analysis
5. Based on the complete model
6. Consider path constraints
Step 1. Get execution traces - Get representative traces efficiently

- Testcase selection
  - Diverse sources: mutator, crawler
  - Select representative ones: min-set coverage analysis based on sub-types (e.g., tags in MP4)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ftyp</td>
<td></td>
<td></td>
<td>✓ file type and compatibility</td>
</tr>
<tr>
<td>pdin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moov</td>
<td></td>
<td></td>
<td>✓ container for all the metadata</td>
</tr>
<tr>
<td>mvhd</td>
<td></td>
<td></td>
<td>✓ movie header, overall declarations</td>
</tr>
<tr>
<td>trak</td>
<td></td>
<td></td>
<td>✓ container for an individual track or stream</td>
</tr>
<tr>
<td>tkhd</td>
<td></td>
<td></td>
<td>✓ track header, overall information about the track</td>
</tr>
<tr>
<td>tref</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>edts</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MP4 file structures
Step 1. Get execution traces - Get representative traces efficiently

- Binary instrument with record and replay

Inspired by PANDA (CCS’ 13)

Our instrument: Xede (RAID’ 15)
Step 2. Identify heap operations - **Handle custom function implementations**

- Custom Heap allocation operations
  - Functionality
    - **A.** return a heap pointer
  - Process inside function
    - **B1.** invoke standard allocation interfaces
    - **B2.** response to allocation applications bases on different sizes
    - **B3.** pad extra bytes to align
    - **B4.** update heap management structures in a critical way
  - Used way
    - **C1.** return space is written before read
    - **C2.** allocator is called many times
    - **C3.** allocator returns different space
    - **C4.** allocator sometimes initials the space
Step 2. Identify heap operations - Handle custom function implementations

- Custom Heap access operations
  - Access the buffer in loops
  
  ```
  .text:78108701  mov  al, [esi]
  .text:78108702  add  esi, 1
  .text:78108703  mov  [edi], al
  .text:78108704  add  edi, 1
  .text:78108705  test  al, al
  .text:78108706  jz   short loc_78108700
  .text:78108707  sub  ebx, 1
  .text:78108708  jnz  short loc_78108703
  ```

  - Access the buffer with REP instructions

  ```
  .text:7855A6F2  mov  al, [esi+3]
  .text:7855A6F3  mov  [edi+3], al
  .text:7855A6F4  mov  al, [esi+2]
  .text:7855A6F5  mov  [edi+2], al
  .text:7855A6F6  mov  eax, [ebp+8]
  .text:7855A6F7  pop   esi
  .text:7855A6F8  pop   edi
  .text:7855A6F9  leave
  .text:7855A6FA  ret
  ```

  - Access the buffer with continuous instructions

  ```
  .text:78108701  mov  al, [esi]
  .text:78108702  add  esi, 1
  .text:78108703  mov  [edi], al
  .text:78108704  add  edi, 1
  .text:78108705  test  al, al
  .text:78108706  jz   short loc_78108700
  .text:78108707  sub  ebx, 1
  .text:78108708  jnz  short loc_78108703
  ```

Identify loops as continuously repeated sub-strings, ending with jump instructions.
Step 3. Track Spatial Attributes - Pair heap operations accurately

- Pair heap access operations to heap allocation operations
  - Pointer taint analysis
    - Taint source: heap pointers returned from allocators
    - Heap access: dereference of a heap pointer
  - Under-taint issue

//EAX is a pointer, EBX is another pointer
//EAX (Taint_{EAX}) EBX (Taint_{EBX})

SUB EAX, EBX
//previous solutions bleach EAX’s taint
MOV ESI, EAX
...
//ESI=(EAX-EBX)+EBX=EAX, not related to EBX
ADD ESI, EBX

Previous solution Signed taint label

EAX (NULL) EAX (Taint_{EAX}, —Taint_{EBX})
ESI (NULL) ESI (Taint_{EAX}, —Taint_{EBX})
... ...
ESI (Taint_{EBX}) ESI (Taint_{EAX})

...
Step 3. Track Taint Attributes - **Use fine-grained taint analysis**

- Multiple label taint analysis at byte level
- Kernel instructions: NOT record

- Sub-Step: Heap Operation Pairs Sorting
  - Select more vulnerable pairs
  - Filter heap management functions
Step 4. Build Vulnerability Conditions - Based on the complete model

- Range_{access} > Range_{obj}
- Consider integer overflow (IO2BO)

Step 5. Generate PoC inputs - Consider path constraints

- Analyze from the last read to the vulnerable point
- Only symbolize relevant input bytes
- Use Z3 to solve the conditions
Evaluation

Effectiveness

- Test on known vulnerabilities
  - Correct the vulnerable points in PoC files
  - Test if we could find the same vulnerabilities
- Discover unknown vulnerabilities
  - Representative samples

**Known vulnerabilities (8 vuls in 8 apps)**

<table>
<thead>
<tr>
<th>ID</th>
<th>Application</th>
<th>version</th>
<th>input</th>
<th>CVE-2010-1932</th>
<th>Xnview</th>
<th>1.97.4</th>
<th>mbm</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2011-5233</td>
<td>irfanview</td>
<td>4.30</td>
<td>tif</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSVDB-83812</td>
<td>ZiplFast</td>
<td>3.0 pro</td>
<td>zip</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVE-2014-1761</td>
<td>Microsoft Word</td>
<td>2010</td>
<td>rtf</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDB-ID-39353</td>
<td>VLC</td>
<td>2.2.1</td>
<td>mp4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDB-ID-17363</td>
<td>1ClickUnzip</td>
<td>3.0.0</td>
<td>zip</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVE-2010-2553</td>
<td>MediaPlayer</td>
<td>9.00.00.4503</td>
<td>avi</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVE-2015-0327</td>
<td>Adobe Flash</td>
<td>13sa</td>
<td>swf</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Unknown vulnerabilities (47 vuls in 17 apps)**

<table>
<thead>
<tr>
<th>ID (count)</th>
<th>Application</th>
<th>version</th>
<th>input</th>
<th>bug status</th>
</tr>
</thead>
<tbody>
<tr>
<td>new (1)</td>
<td>Feiq</td>
<td>3.0.0.2</td>
<td>tcp</td>
<td>reported</td>
</tr>
<tr>
<td>new (1)</td>
<td>WMPlayer</td>
<td>12.0.7601</td>
<td>mp4</td>
<td>reported</td>
</tr>
<tr>
<td>new (3)</td>
<td>VLC</td>
<td>2.2.1</td>
<td>mp4</td>
<td>fixed</td>
</tr>
<tr>
<td>new (1)</td>
<td>VLC</td>
<td>2.2.4</td>
<td>mp4</td>
<td>reported</td>
</tr>
<tr>
<td>new (2)</td>
<td>iTunes</td>
<td>12.4.3.1</td>
<td>mp4</td>
<td>reviewing</td>
</tr>
<tr>
<td>new (1)</td>
<td>ffmpeg</td>
<td>c0cb53c</td>
<td>mp4</td>
<td>CVE</td>
</tr>
<tr>
<td>new (6)</td>
<td>QQPlayer</td>
<td>3.9(936)</td>
<td>mp4</td>
<td>rewarded</td>
</tr>
<tr>
<td>new (1)</td>
<td>QQMusic</td>
<td>11.5</td>
<td>m4a</td>
<td>rewarded</td>
</tr>
<tr>
<td>new (1)</td>
<td>BaiduPlayer</td>
<td>5.2.1.3</td>
<td>mp4</td>
<td>reviewing</td>
</tr>
<tr>
<td>new (2)</td>
<td>RealPlayer</td>
<td>16.0.6.2</td>
<td>mp4</td>
<td>CVE</td>
</tr>
<tr>
<td>new (1)</td>
<td>MPlayer</td>
<td>r37802</td>
<td>mp4</td>
<td>reported</td>
</tr>
<tr>
<td>new (3)</td>
<td>KMPlayer</td>
<td>3.9.1.138</td>
<td>mp4</td>
<td>fixed</td>
</tr>
<tr>
<td>new (4)</td>
<td>KMPlayer</td>
<td>4.1.1.5</td>
<td>mp4</td>
<td>reported</td>
</tr>
<tr>
<td>new (7)</td>
<td>Potplayer</td>
<td>1.6.60136</td>
<td>mp4</td>
<td>fixed</td>
</tr>
<tr>
<td>new (2)</td>
<td>Potplayer</td>
<td>1.6.62949</td>
<td>mp4</td>
<td>reported</td>
</tr>
<tr>
<td>new (5)</td>
<td>Splplayer</td>
<td>3.7</td>
<td>mp4</td>
<td>reported</td>
</tr>
<tr>
<td>new (2)</td>
<td>MS Word</td>
<td>2007.10.16</td>
<td>rtf</td>
<td>reviewing</td>
</tr>
<tr>
<td>new (1)</td>
<td>WPS Word</td>
<td>10.10.5803</td>
<td>doc</td>
<td>reported</td>
</tr>
<tr>
<td>new (2)</td>
<td>OpenOffice</td>
<td>4.1.2</td>
<td>doc</td>
<td>reviewing</td>
</tr>
<tr>
<td>new (1)</td>
<td>IrfanView</td>
<td>4.41</td>
<td>m3u</td>
<td>fixed</td>
</tr>
</tbody>
</table>
Evaluation

Details

• We have discovered 6 vulnerabilities of 8 known ones.
• CVE-2015-0327: overwrite the program function with scripts and change the behavior
• CVE-2010-2553: multiple level nested loops of accesses

<table>
<thead>
<tr>
<th>ID</th>
<th>record-replay phase</th>
<th>analysis phase</th>
<th>resolve phase</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>snapshot size</td>
<td>changelog size</td>
<td>record time</td>
</tr>
<tr>
<td>CVE-2010-1932</td>
<td>430.6MB</td>
<td>36.6MB</td>
<td>29s</td>
</tr>
<tr>
<td>CVE-2011-5233</td>
<td>516.1MB</td>
<td>18.5MB</td>
<td>37s</td>
</tr>
<tr>
<td>OSVDB-83812</td>
<td>819.3MB</td>
<td>13.6MB</td>
<td>83s</td>
</tr>
<tr>
<td>CVE-2014-1761</td>
<td>855.3MB</td>
<td>52.3MB</td>
<td>178s</td>
</tr>
<tr>
<td>EDB-ID-39353</td>
<td>507.6MB</td>
<td>15.0MB</td>
<td>62s</td>
</tr>
<tr>
<td>EDB-ID-17363</td>
<td>500.2MB</td>
<td>32.6MB</td>
<td>70s</td>
</tr>
<tr>
<td>CVE-2010-2553</td>
<td>282.5MB</td>
<td>22.9MB</td>
<td>100s</td>
</tr>
<tr>
<td>CVE-2015-0327</td>
<td>610.8MB</td>
<td>13.8MB</td>
<td>34s</td>
</tr>
</tbody>
</table>
**Evaluation**

**Case studies**

- Tainted access offset
  - Offset is influenced by input

  ```c
  dst address = Object + offset
  memcpy(Dst, Src)
  ```

- Implicit taint situations
  - Allocation side: Size of malloc is based on the length of inputs
  - Access side: Number of data copy is based on keywords

```c
while (input != '\0') {
    length++;
}
buffer = malloc(length);
```

```c
while (input == keywords) {
    memcpy();
}
```
Case studies

• Multiple vulnerabilities in one trace
  – We found two vulnerable points when testing the PoC of CVE-2014-1761

• Long testing time
  – A VLC vulnerability happened at the end of playing for more than several minutes
Evaluation

Comparison

• WinAFL
  – Need more knowledge of the target software and its functions
  – Fail to find new vulnerabilities

• Radamsa
  – Black-box fuzzer with various kinds of heuristics and change patterns
  – Potplayer: 1144 crashes, 11 crash points, 3 vulnerable points (included by HOTracer)
Future Work

- Custom heap management functions
- Complicated heap access operations
- Constraint simplification and solve
- Exploitability assess
- Use after free
Conclusion

- We proposed a new offline dynamic analysis solution to discover heap overflow vulnerabilities.
- We pointed out the root cause of heap overflow vulnerabilities is the inconsistency between heap operations.
- We addressed several challenges to make the solution practical and efficient.
- We implemented a prototype system, HOTracer, and found 47 previously unknown heap vulnerabilities in 17 applications with it.
Thank you

Xiangkun Jia
jiaxiangkun@tca.iscas.ac.cn