Playing for K(H)eaps:
Understanding and Improving Linux Kernel Exploit Reliability

Kyle Zeng*, Yueqi Chen*, Haehyun Cho, Xinyu Xing, Adam Doupé, Yan Shoshitaishvili, Tiffany Bao

1Arizona State University
2Pennsylvania State University
3University of Colorado Boulder
4Soongsil University
5Northwestern University

*First two authors contributed equally to this work
Linux Kernel Heap Exploit
Linux kernel exploits are dangerous

Linux kernel exploits are known to be unreliable

Exploit stabilization heavily relies on personal expertise
Systematically study why Linux kernel heap-based exploits are unreliable
Research Questions

• What are the commonly used exploit stabilization techniques?
• How effective are existing techniques?
• Why do existing techniques work?
• Is there any way to further improve exploit reliability?
Our Approach

Qualitative Interview → Techniques → Quantitative Experiment → Results → Investigation

Modeling

New Technique → Model → Combination

Knowledge
Technique Collection

11 Linux kernel security experts

Defragmentation
Heap Grooming
Single-Thread Heap Spray
Multi-Process Heap Spray
CPU Pinning

*obtained exemption from IRB
Quantitative Experiment

- Real-world exploits: 17 public exploits for distinct CVEs
- Baseline exploits: strip away existing techniques
- Exploit variants: apply one single technique to baseline

85 samples in total
## Quantitative Experiment Result

<table>
<thead>
<tr>
<th>Success</th>
<th>Baseline</th>
<th>Defragment</th>
<th>Heap Grooming</th>
<th>Single-Thread Spray</th>
<th>Multi-Process Spray</th>
<th>CPU Pinning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>38.61%</td>
<td>31.88%</td>
<td>74.40%</td>
<td>61.83%</td>
<td>82.55%</td>
<td>51.51%</td>
</tr>
</tbody>
</table>

Evaluation result of all techniques
Quantitative Experiment Result – Cont.

<table>
<thead>
<tr>
<th>Success</th>
<th>Baseline</th>
<th>Defragmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>13.05%</td>
<td>42.64%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Success</th>
<th>Baseline</th>
<th>Defragmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>49.26%</td>
<td>27.40%</td>
</tr>
</tbody>
</table>

Evaluation Result for OOB Exploits

Evaluation Result for non-OOB Exploits
Kernel Heap Exploit Model

Context Setup  
Vulnerability Effect Delay  
Allocator Bracing  
Final Preparation

Start  
Vulnerability Triggered  
Allocator Corrupted  
Allocator Braced  
Payload Triggered  
Object Release  
Heap Layout Preparation  
Heap Layout Preparation  
Dangling Pointer Created  
Freelist Overwritten  
Object Overflowed

OOB-Object Exploits  
UAF/DF Exploits  
OOB-Freelist Exploits
Critical Phases

Slot-Critical Phase

Allocator-Critical Phase

OOB Exploits

DF Exploits
Unreliability Factors

• Unknown Heap Layout

• Unexpected Heap Usage

• Unwanted Task Migration

• Unpredictable Corruption Timing
Kernel Heap Exploit Model

Context Setup
- Vulnerability Triggered
- Dangling Pointer Created
- Heap Layout Preparation
- Slot-Critical

Vulnerability Effect Delay
- Allocator Corrupted
- Object Release Take Effect
- Object Overflowed

Allocator Bracing
- Freelist Overwritten
- Allocator Braced

Final Preparation
- Payload Triggered
- Slot-Critical & Allocator-Critical

OOB-Object Exploits

OOB-Freelist Exploits
- Slot-Critical
- Allocator-Critical
Context Conservation

Critical Phase

- alloc vuln obj
- ?
- alloc victim
- do overflow

context switch
reschedule

Other Processes

OOB Exploits
Use Time Stamp Counter (TSC) as the context-switch indicator

\[
\begin{align*}
tsc1 &= \text{rdtsc()} \\
tsc2 &= \text{rdtsc()} \\
diff &= tsc2 - tsc1
\end{align*}
\]

If diff is huge, then it is a fresh time slice

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Context Conservation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>62.48%</td>
<td>64.07%</td>
</tr>
<tr>
<td>Busy</td>
<td>36.75%</td>
<td>49.84%</td>
</tr>
</tbody>
</table>
Combo Technique

• Unknown Heap Layout       ← Defragmentation
• Unexpected Heap Usage      ← Context Conservation
• Unwanted Task Migration    ← Multi-Process Heap Spray
• Unpredictable Corruption Timing ← CPU Pinning

What if we combine them?
Combo Technique – Cont.

Exploit Variant: baseline + applicable techniques

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Real-world</th>
<th>Combo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>36.51%</td>
<td>66.99%</td>
<td>91.15%</td>
</tr>
</tbody>
</table>

Evaluation Result
Conclusion

• Systematically studied the kernel heap exploit reliability problem
• Proposed a model to explain the problem and guide future research
• Discovered a new technique that improve exploit reliability by 14.87%
• Designed a technique combination that improves exploit reliability by 135.53%
Playing for K(H)eaps:
Understanding and Improving Linux Kernel Exploit Reliability

Thank you!
Q & A

https://github.com/sefcom/KHeaps

Kyle Zeng
zengyhkyle@asu.edu
@ky1ebot
@Kyle-Kyle