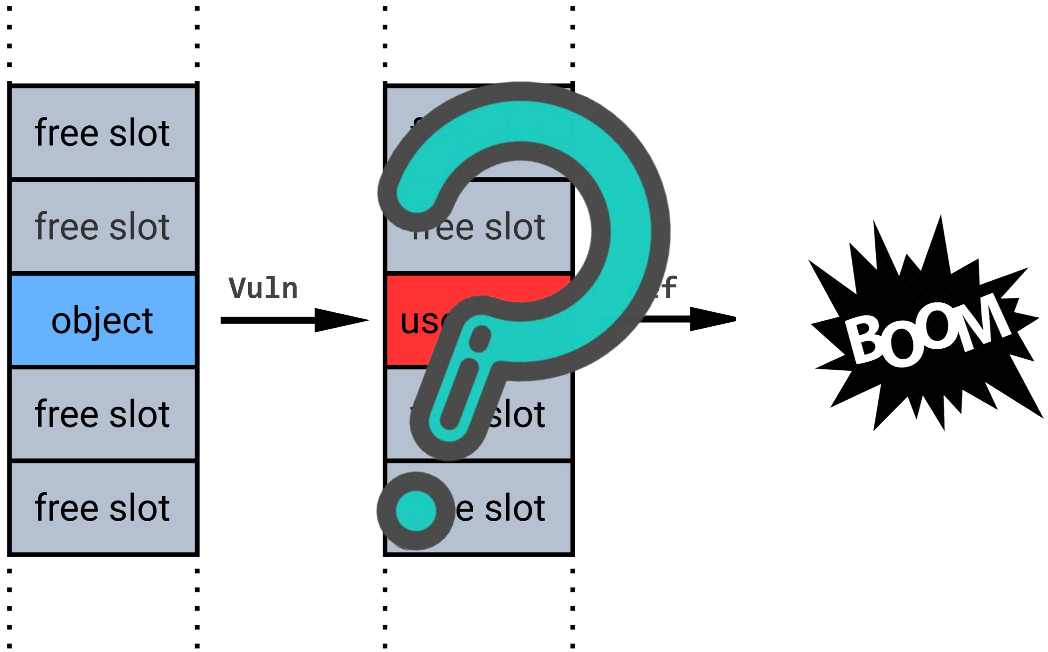
Playing for K(H)eaps:
Understanding and Improving Linux Kernel Exploit Reliability

Kyle Zeng[*1], Yueqi Chen[*2,3], Haehyun Cho[1,4], Xinyu Xing[2,5], Adam Doupé[1], Yan Shoshitaishvili[1], Tiffany Bao[1]

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

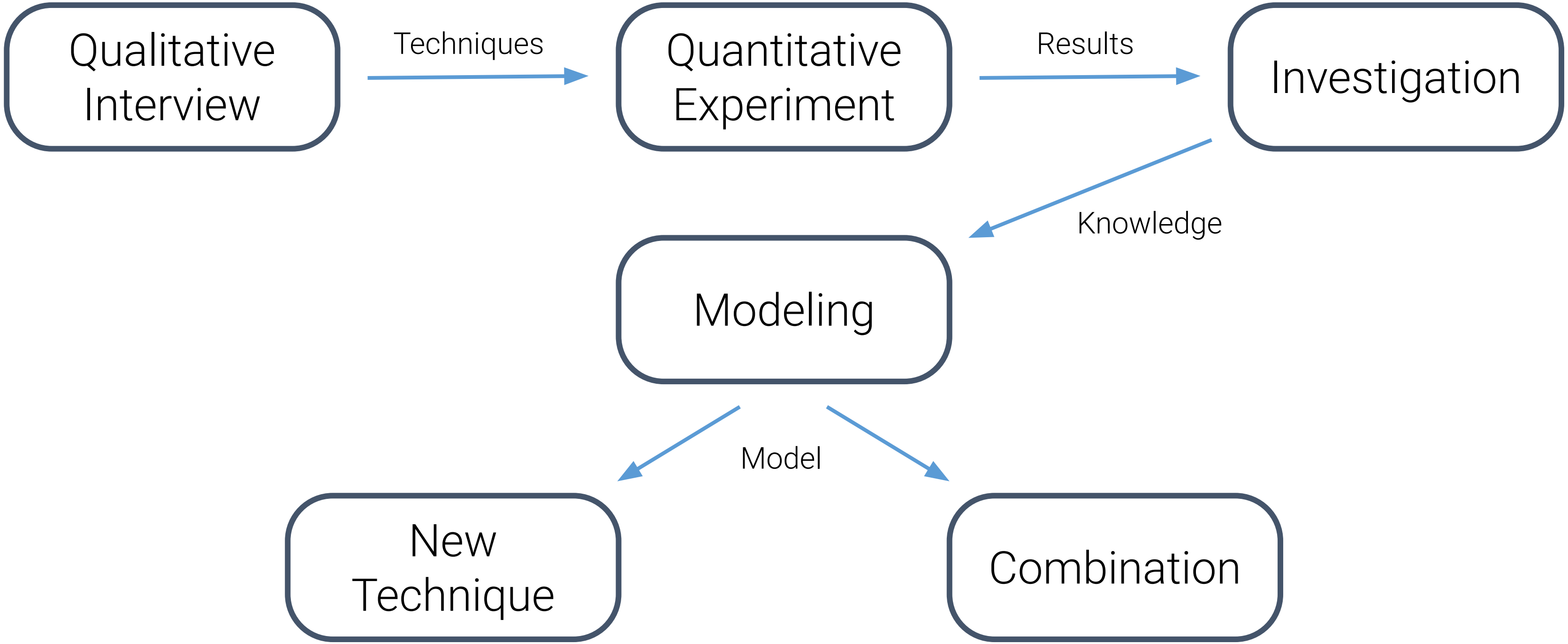
Exploit stabilization heavily relies on personal expertise

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



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

	Baseline	Defragment	Heap Grooming	Single-Thread Spray	Multi-Process Spray	CPU Pinning
Success	38.61%	31.88%	74.40%	61.83%	82.55%	51.51%

Evaluation result of all techniques

Quantitative Experiment Result – Cont.

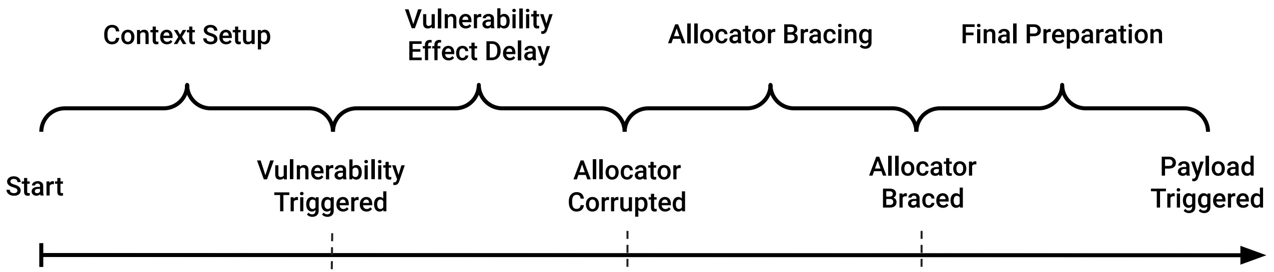
	Baseline	Defragmentation
Success	13.05%	42.64%

Evaluation Result for OOB Exploits

	Baseline	Defragmentation
Success	49.26%	27.40%

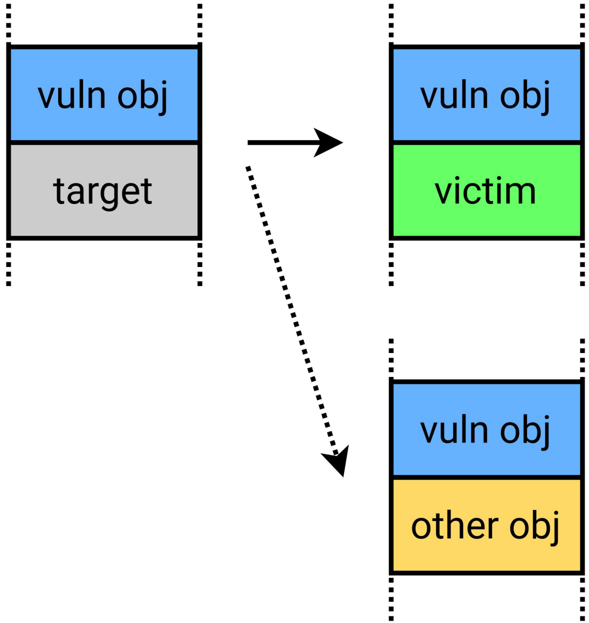
Evaluation Result for non-OOB Exploits

Kernel Heap Exploit Model



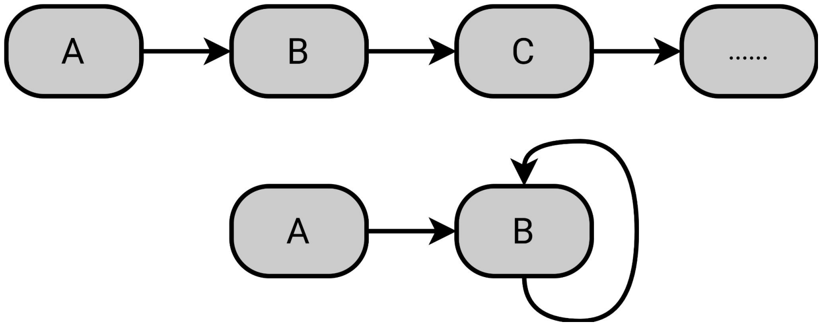
Critical Phases

Slot-Critical Phase



OOB Exploits

Allocator-Critical Phase

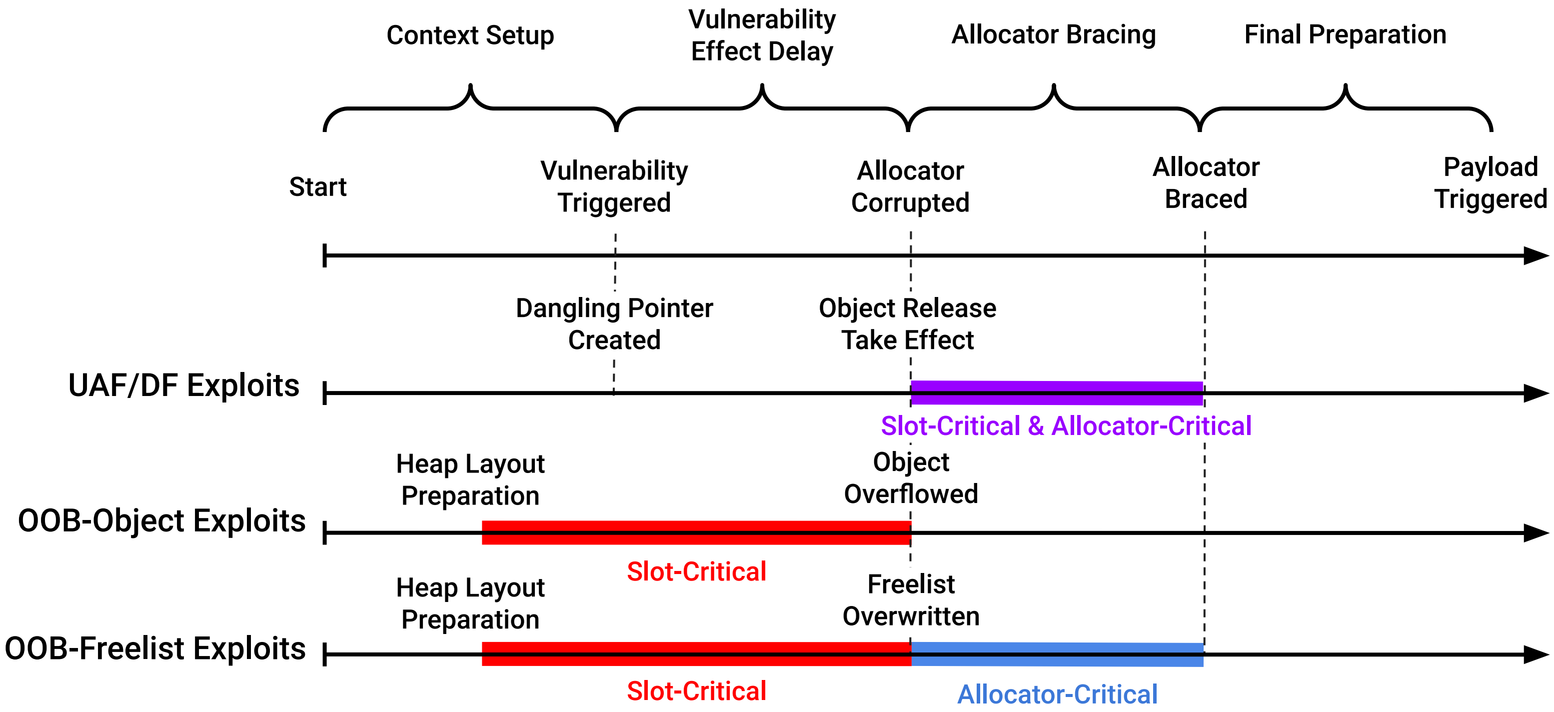


DF Exploits

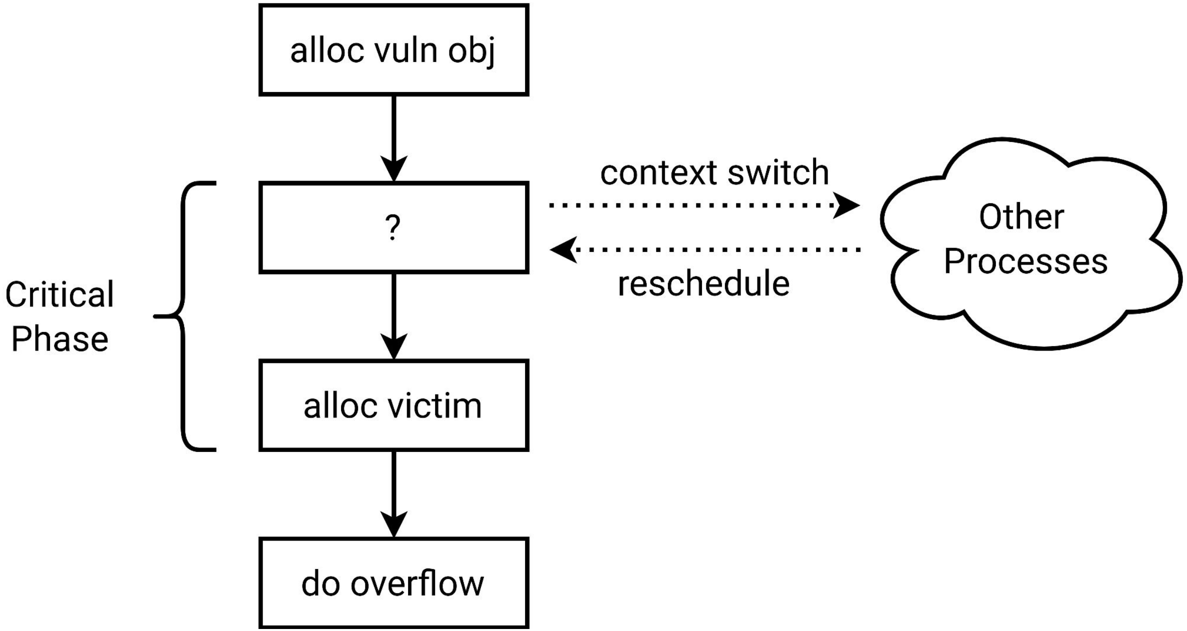
Unreliability Factors

- Unknown Heap Layout
- Unexpected Heap Usage
- Unwanted Task Migration
- Unpredictable Corruption Timing

Kernel Heap Exploit Model



Context Conservation



OOB Exploits

Context Conservation – Cont.

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

```
tsc1 = rdtsc()
tsc2 = rdtsc()
diff = tsc2 - tsc1
```

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

	Baseline	**Context Conservation**
Idle	62.48%	64.07%
Busy	36.75%	49.84%

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

	Baseline	Real-world	Combo
Success	36.51%	66.99%	91.15%

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

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Kyle Zeng

zengyhkyle@asu.edu

@ky1ebot

GitHub @Kyle-Kyle