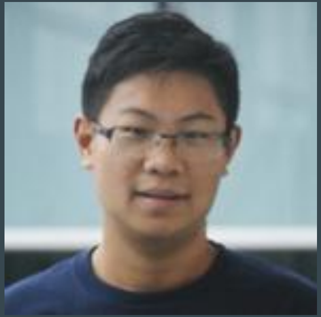
Facilitate Linux Kernel Exploitation Step by Step

Yueqi (Lewis) Chen
The Pennsylvania State University

Who am I?



Yueqi Chen @Lewis_Chen_

- 3rd year Ph.D. student at Pennsylvania State
- interested in OS security and vulnerability analysis
- looking for 2020 summer internship
- I have a story to share

How I began my “career” in Linux kernel exploitation?

About three years ago, I received my bachelor degree and went to the U.S. for Ph.D. study. I was a noob and knew very little about security at that moment.

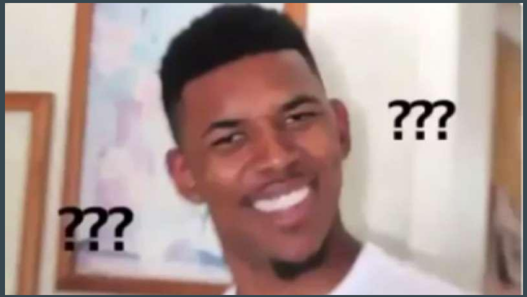
Me: “What can I do?”

Advisor: “Hey, Linux kernel is vulnerable. Do you know how to exploit them?”

Me: “Emmmmm, frankly, I don’t know.”

Advisor: “Then learn it.”

Me; “What?”



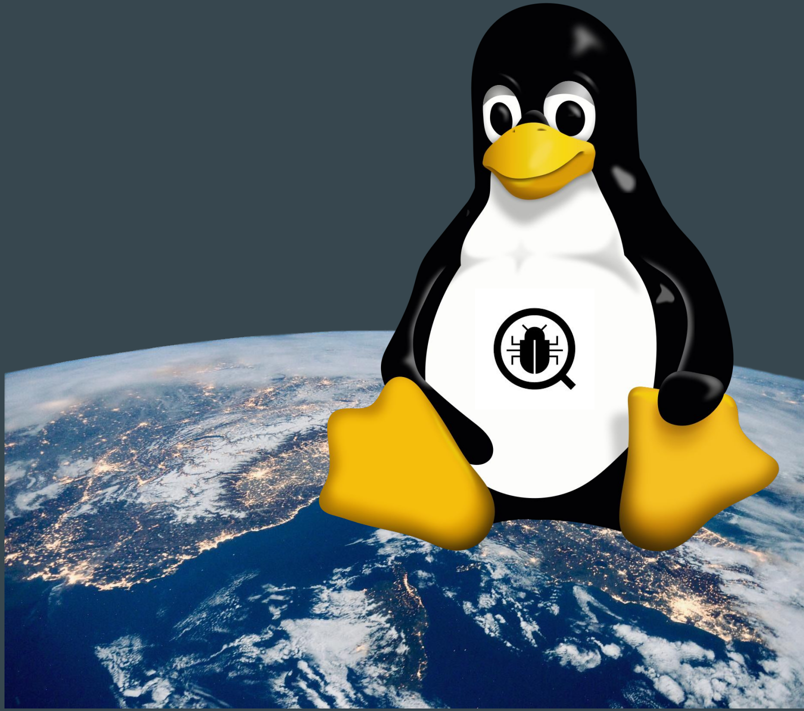
I learned two facts about Linux

"Civilization runs on Linux" [1][2]

- Android (2e9 users), cloud servers, desktops
- cars, transportation
- nuclear submarines, etc.

Linux kernel is buggy

- 801 CVEs in three years (2017, 2018, 2019)
- 4100+ official bug fixes in 2017
- Syzbot[3] reports nearly 200 bugs/month



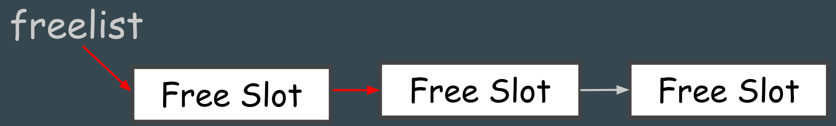
[1] SLTS project, https://lwn.net/Articles/749530/
[2] "Syzbot and the Tale of Thousand Kernel Bugs" - Dmitry Vyukov, Google
[3] syzbot https://syzkaller.appspot.com/upstream

One of the common attack targets is SLAB/SLUB allocator

A single list organizes free slots

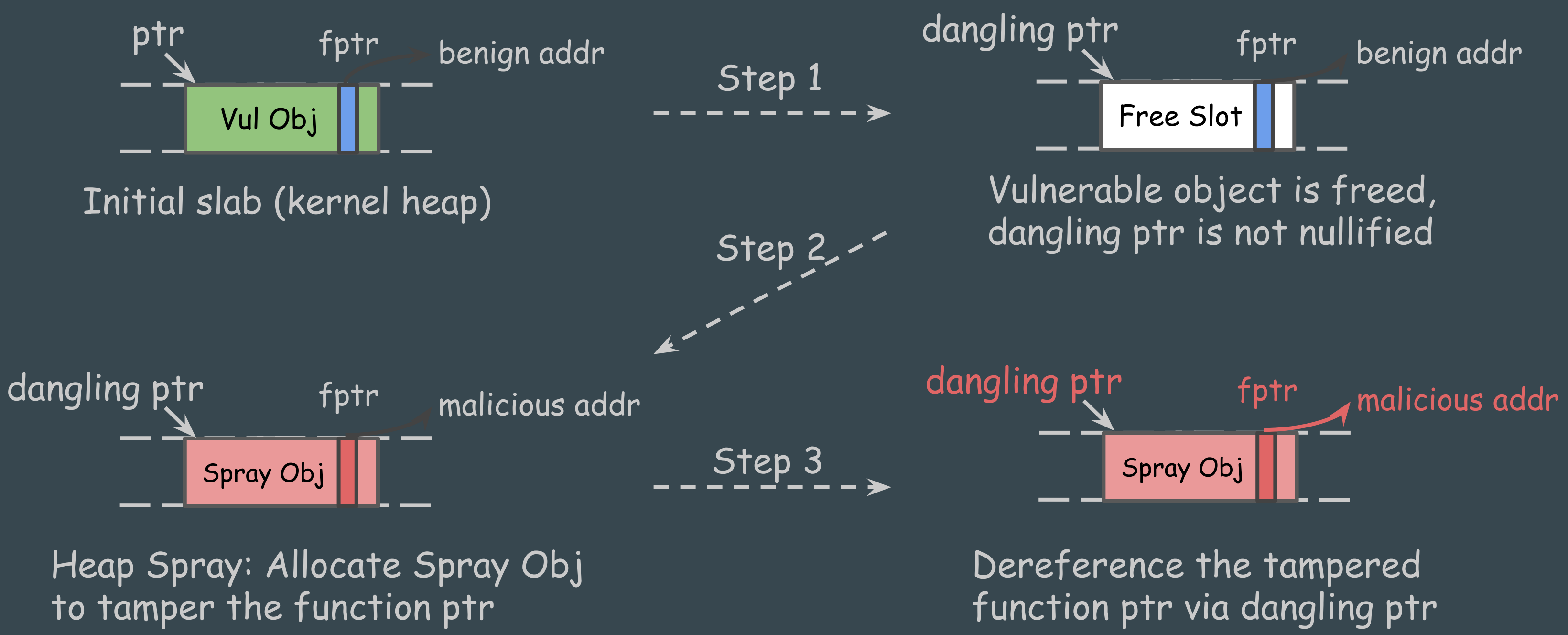
Allocation:
retrieve from the freelist head

Deallocation:
recycle to the freelist head



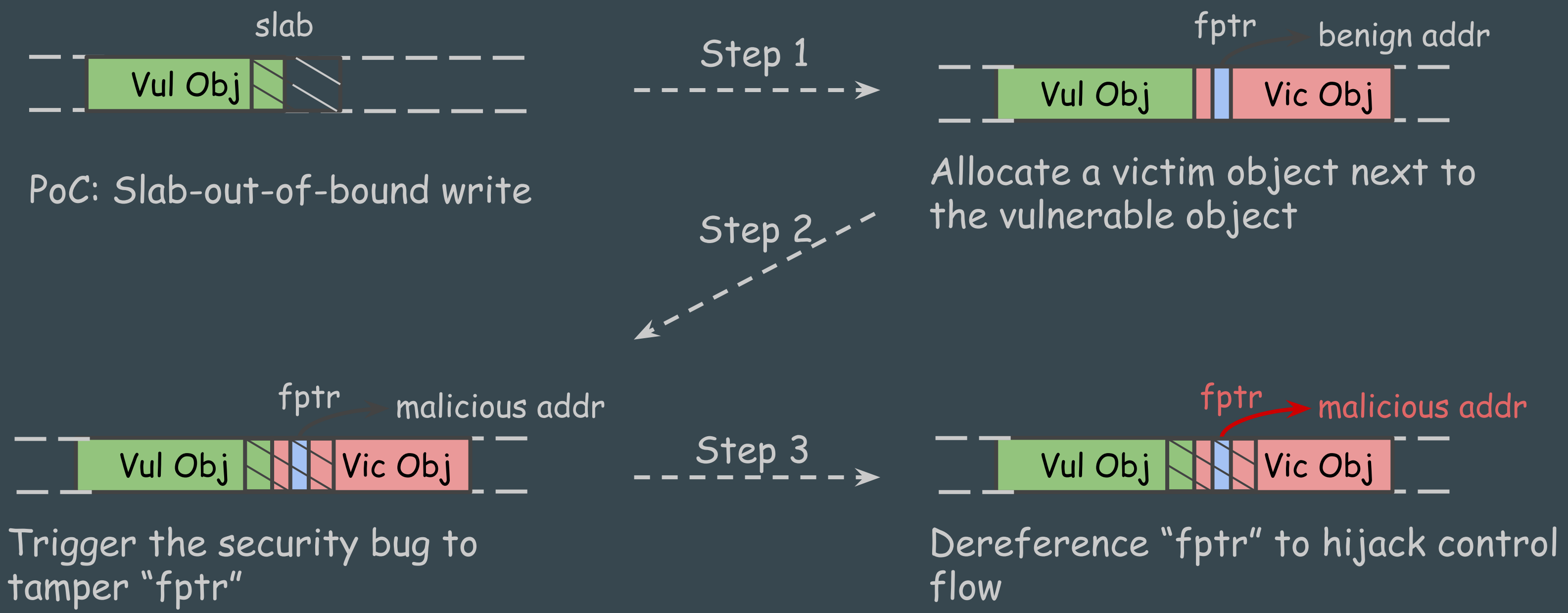
Highly simplified, not entirely correct

I read writeups and debugged public exploits



Exploit A Use-After-Free in Three Steps

I read writeups and debugged public exploits (cont.)



Exploit A Slab Out-of-bound Write in Three Steps

After months, I went back to my advisor

Me: “Now I know how to exploit Linux kernel vulnerabilities.”

Advisor: “Good job.”

Me: “But I find it’s still challenging to craft an exploit for a new vulnerability..”

Advisor: “Tell me more?”

Me: “The first challenge is … ”

Challenge 1: how to corrupt “correctly”?

For use-after-free vulnerabilities, a Proof-of-Concept (PoC) program dereferences a non-critical variable in freed object. For example

```
freed_obj->cnt++; // a normal counter, not reference count
```

But I want a dereference like this

```
freed_obj->op(xx, yy); // indirect call, control-flow hijacking!!!
```

Challenge 2: which objects to use for Fengshui/Spraying

I have a slab out-of-bound write which can write controllable 12 bytes to the next object. Which object to overwrite?

- Common candidates: struct file, struct tty_struct, etc.

I have a use-after-free which dereference critical data. Which object for heap spraying?

- Common candidates: send{m}msg, add_key, etc.

However,

1. common candidates don't match with the vulnerability.
2. fengshui/spraying is hard due to side-effect.

Challenge 3: bypass mitigations in general approach

Kernel is exploited for many years. Many mitigations have been built into the kernel.

1. SMAP/SMEP/PAN
2. KASLR
3. Non-executable Physmap
4. etc.

I need to specify the way to bypass above mitigations case by case.

Is there a general approach?

After months, I went back to my advisor (cont.)

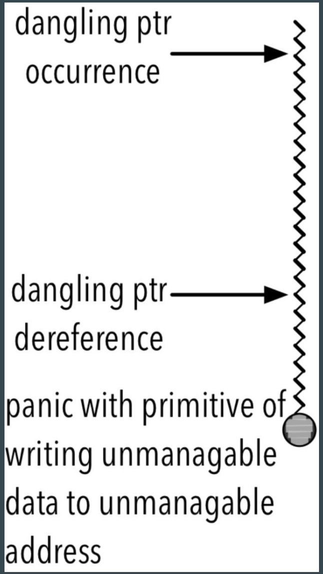
Advisor: “Sounds interesting. Could you solve them?”

Me: “Are you serious?”

Advisor: “Yes.”

Me: “ OK, I will make a try”

Try 1: Challenge Analysis



Free

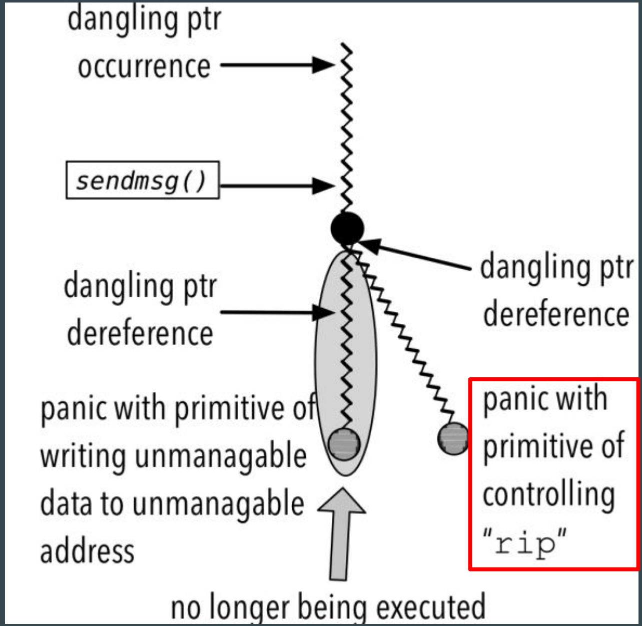
Use

Panic

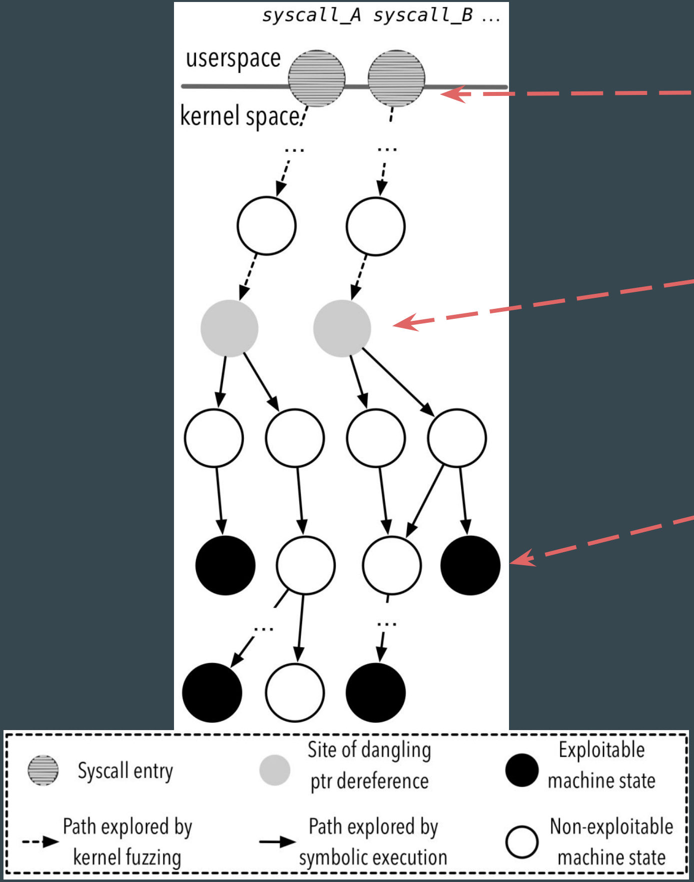
Free

Heap spray

New Use



Try 1: Solution



1. Kick in kernel fuzzing to explore new use sites after freeing the vulnerable object
2. Symbolically execute the kernel from the new use sites to check if useful primitives (e.g., RIP control, arbitrary read/write) can be obtained
3. Solve conjunction of path constraints towards primitives and constraints for primitives (e.g., function pointer == the malicious address) to calculate the content of spray object

Try 1: Results

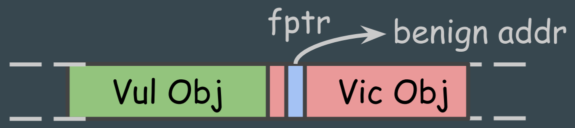
- 15 kernel UAF vulnerabilities as evaluation set
- Escalated exploitability of 7 vulnerabilities
- The new use sites found generate 12 additional exploits bypassing SMEP and 3 additional exploits bypassing SMAP
- Example: CVE-2017-15649

CVE-ID	*# of public exploits*		*# of generated exploits*	
	SMEP	SMAP	SMEP	SMAP
2017-17053	0	0	1	0
2017-15649	0	0	3	2
2017-15265	0	0	0	0
2017-10661	0	0	2	0
2017-8890	1	0	1	0
2017-8824	0	0	2	2
2017-7374	0	0	0	0
2016-10150	0	0	1	0
2016-8655	1	1	1	1
2016-7117	0	0	0	0
2016-4557	1	1	4	0
2016-0728	1	0	3	0
2015-3636	0	0	0	0
2014-2851	1	0	1	0
2013-7446	0	0	0	0
Overall	5	2	19	5

Table 4: Exploitability comparison with and without FUZE.

Try 2: Challenge Analysis

1. Which kernel object is useful for exploitation
   - similar size/same type to be allocated to the same cache as the vulnerable object
   - e.g, enclose ptr whose offset is within corruption range

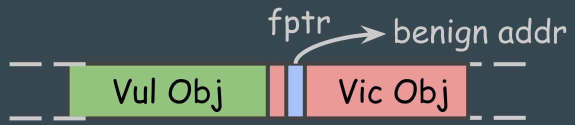


Allocate a victim object next to the vulnerable object

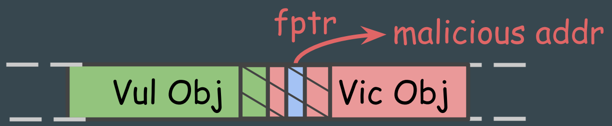
Try 2: Challenge Analysis

1. Which kernel object is useful for exploitation

2. How to (de)allocate and dereference useful objects
   - System call sequence, arguments



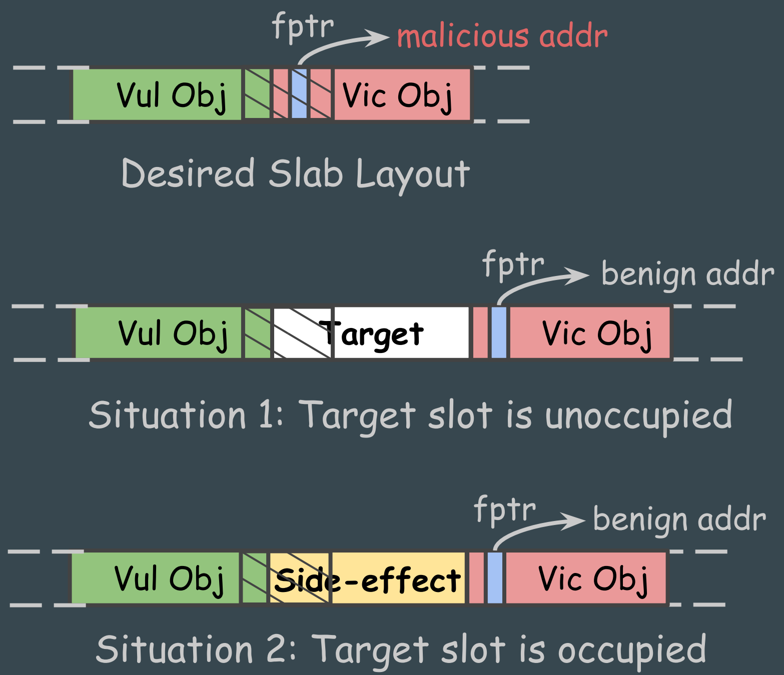
Allocate a victim object next to the vulnerable object



Dereference "fptr" to hijack control flow

Try 2: Challenge Analysis

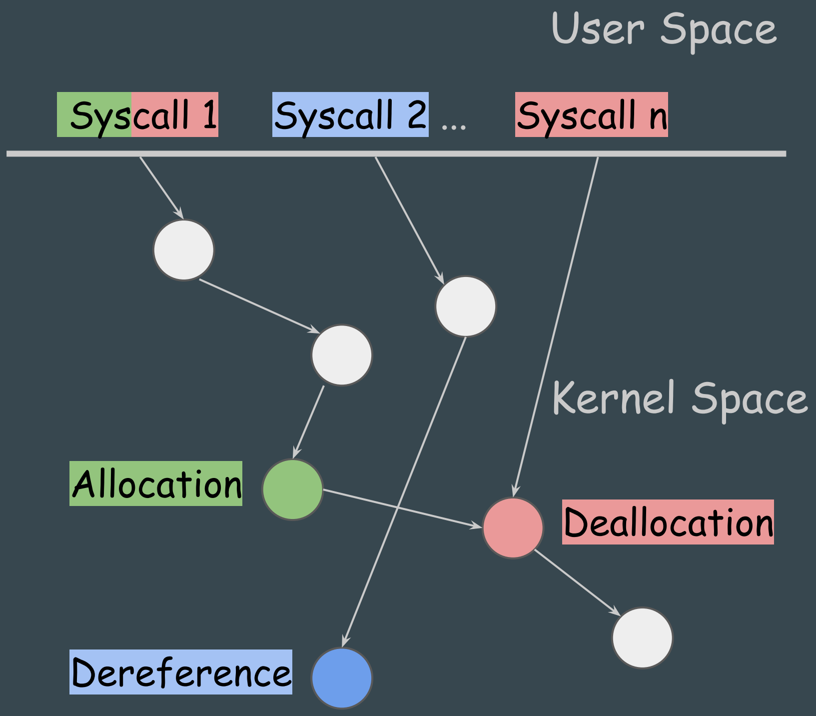
1. Which kernel object is useful for exploitation
2. How to (de)allocate and dereference useful objects
3. How to manipulate slab to reach desired layout
   - unexpected (de)allocation along with vulnerable/victim object makes side-effect to slab layout



Try 2: Solution

build a kernel object database

- Static Analysis to identify useful objects, sites of interest (allocation, deallocation, dereference), potential system calls
- Fuzzing Kernel to confirm System calls and complete arguments

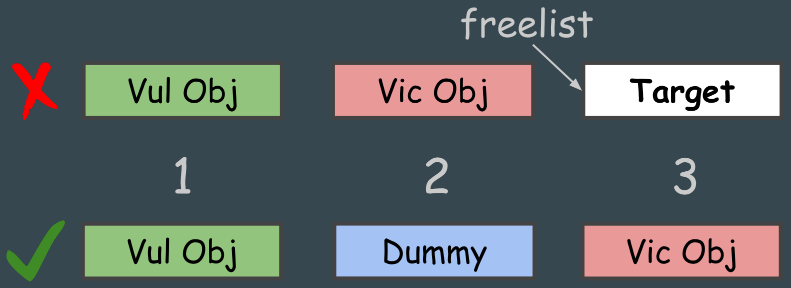


Kernel Call Graph

Try 2: Solution (cont.)

Situation 1: Target slot is unoccupied

- 2 allocations while the order of target slot is 3rd
- add one more allocation of Dummy before Vic Obj



Situation 2: Target slot is occupied

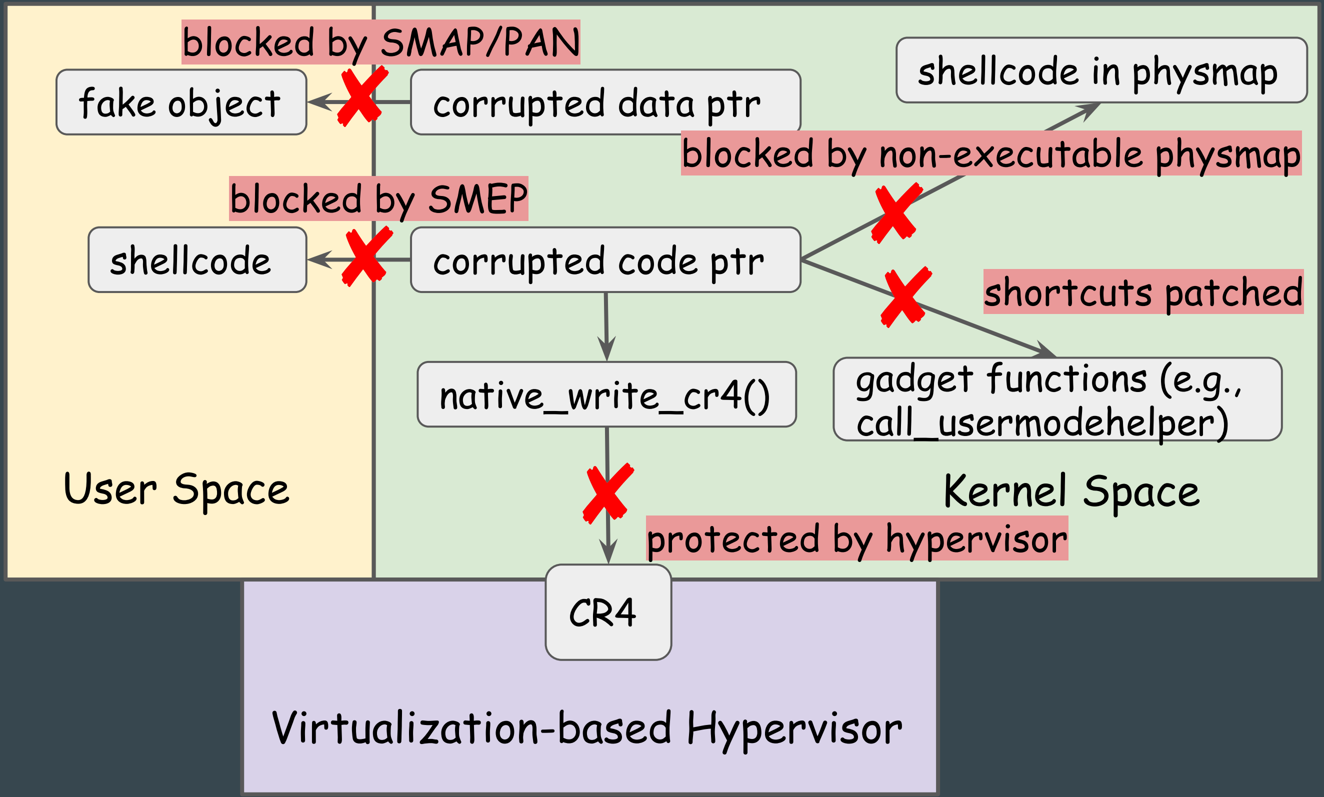
- side-effect object possesses the target
- switch the order of slots holding S-E Obj and Vic Obj in the freelist

Try 2: Results

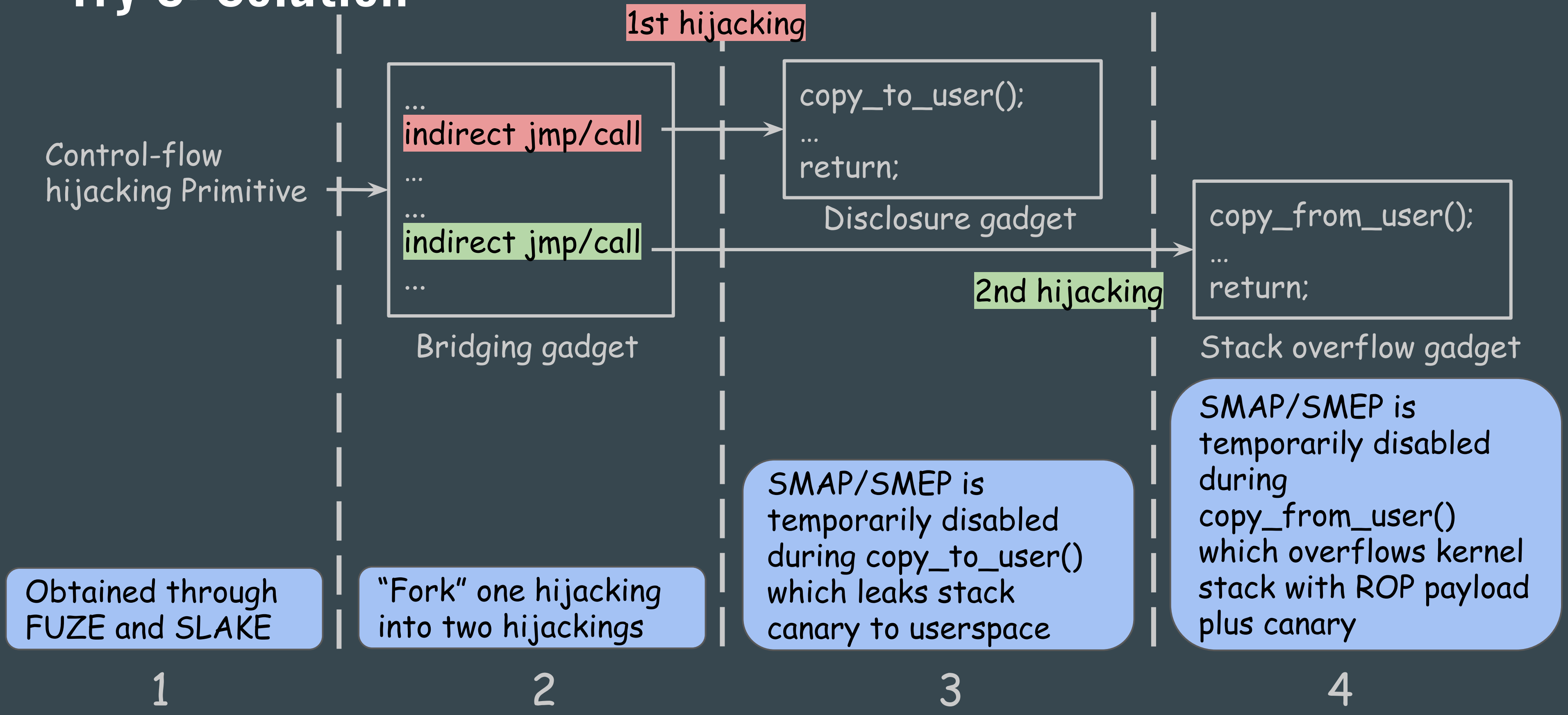
- 27 kernel vulnerabilities, including UAF, Double Free, OOB
- Obtained control-flow hijacking primitive in 14 cases with public exploits and 3 cases without public exploits.

CVE-ID	Type	Exploitation Methods			
		I	II	III	IV
N/A[47]	OOB	5 ($1^{\star}$)	-	-	5 (0)
2010-2959	OOB	13 ($1^{\star}$)	-	-	13 (0)
2018-6555	UAF	-	1($1^{\star}$)	-	-
2017-1000112	OOB	0 (1)	-	-	-
2017-2636	double free	-	0 (1)	-	-
2014-2851	UAF	-	0 (1)	-	-
2015-3636	UAF	-	3 (1)	-	2 (0)
2016-0728	UAF	-	3 (1)	-	4 (0)
2016-10150	UAF	-	3 (1)	-	-
2016-4557	UAF	-	2 (0)	-	-
2016-6187	OOB	-	-	-	6 (1)
2016-8655	UAF	-	3 (1)	-	-
2017-10661	UAF	-	3 (1)	-	-
2017-15649	UAF	-	3 (1)	-	-
2017-17052	UAF	-	0 (0)	-	-
2017-17053	double free	-	-	-	2 (1)
2017-6074	double free	-	3 (1)	12 (0)	-
2017-7184	OOB	10 (0)	-	-	10 (0)
2017-7308	OOB	14 (1)	-	-	14 (0)
2017-8824	UAF	-	3 (1)	-	-
2017-8890	double free	-	4 (1)	4 (0)	-
2018-10840	OOB	0 (0)	-	-	-
2018-12714	OOB	0 (0)	-	-	-
2018-16880	OOB	0 (0)	-	-	-
2018-17182	UAF	-	0 (0)	-	-
2018-18559	UAF	-	3(0)	-	-
2018-5703	OOB	0 (0)	-	-	-

Try 3: Challenge Analysis



Try 3: Solution



Try 3: Results

- 16 CVEs + 3 CTF challenges as evaluation set
- Bypassed mitigations using control-flow hijacking primitives in 17 vulnerabilities

ID	Vulnerability type	Public exploit	KEPLER
CVE-2017-16995	OOB readwrite	✓†	✓
CVE-2017-15649	use-after-free	✓	✓
CVE-2017-10661	use-after-free	✗	✓
CVE-2017-8890	use-after-free	✗	✓
CVE-2017-8824	use-after-free	✓	✓
CVE-2017-7308	heap overflow	✓	✓
CVE-2017-7184	heap overflow	✓	✓
CVE-2017-6074	double-free	✓	✓
CVE-2017-5123	OOB write	✓†	✓
CVE-2017-2636	double-free	✗	✓
CVE-2016-10150	use-after-free	✗	✓
CVE-2016-8655	use-after-free	✓†	✓
CVE-2016-6187	heap overflow	✗	✓
CVE-2016-4557	use-after-free	✗	✓
CVE-2017-17053	use-after-free	✗	✗
CVE-2016-9793	integer overflow	✗	✗
TCTF-credjar	use-after-free	✓†	✓
0CTF-knote	uninitialized use	✗	✓
CSAW-stringIPC	OOB read&write	✓†	✓

End of my story

Me: “I made attempts. And you see these results.”

Advisor: “Looks awesome. What’s your next plan?”

Me: “I kind of know how to proceed this direction. I would like to propose … ”

Advisor: “Well. Next time we meet. We can discuss your proposal examination.”

Thank You

Contact

Twitter: @Lewis_Chen_
Email: ychen@ist.psu.edu
Personal Page: http://www.personal.psu.edu/yxc431/