YONSEI UNIVERSITY
INFORMATION SECURITY & AI SECUTIY LABORATORY

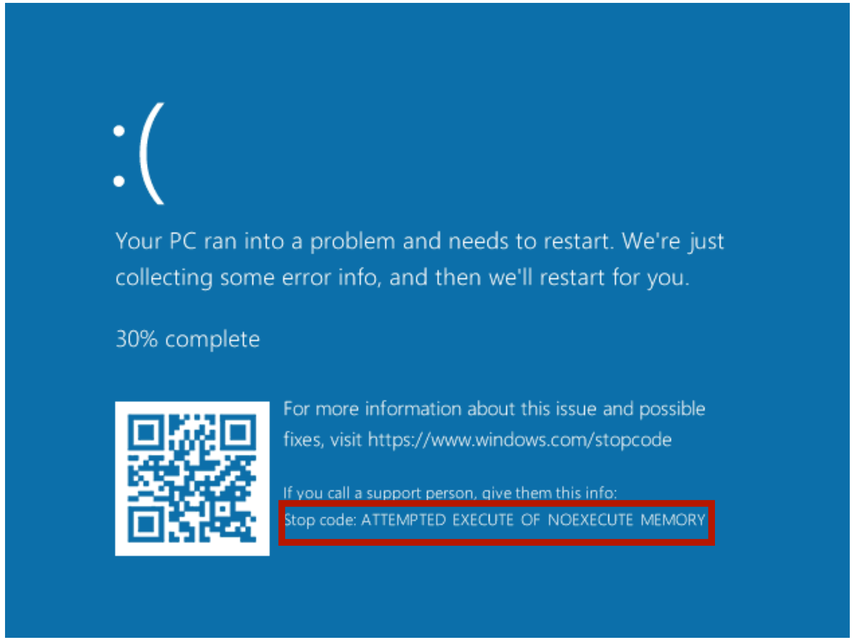
BoKASAN: Binary-only Kernel Address Sanitizer for Effective Kernel Fuzzing

Mingi Cho[1,2], Dohyeon An[1,3], Hoyong Jin[1,4], Taekyoung Kwon[1]

[1] Yonsei University

[2] Theori Inc.
[3] PiLab Technology Inc.
[4] AutoCrypt Inc.

Kernel Bugs are Critical



Researchers Uncover New Linux Kernel 'StackRot' Privilege Escalation Vulnerability

Jul 06, 2023 Ravie Lakshmanan Linux / Endpoint Security

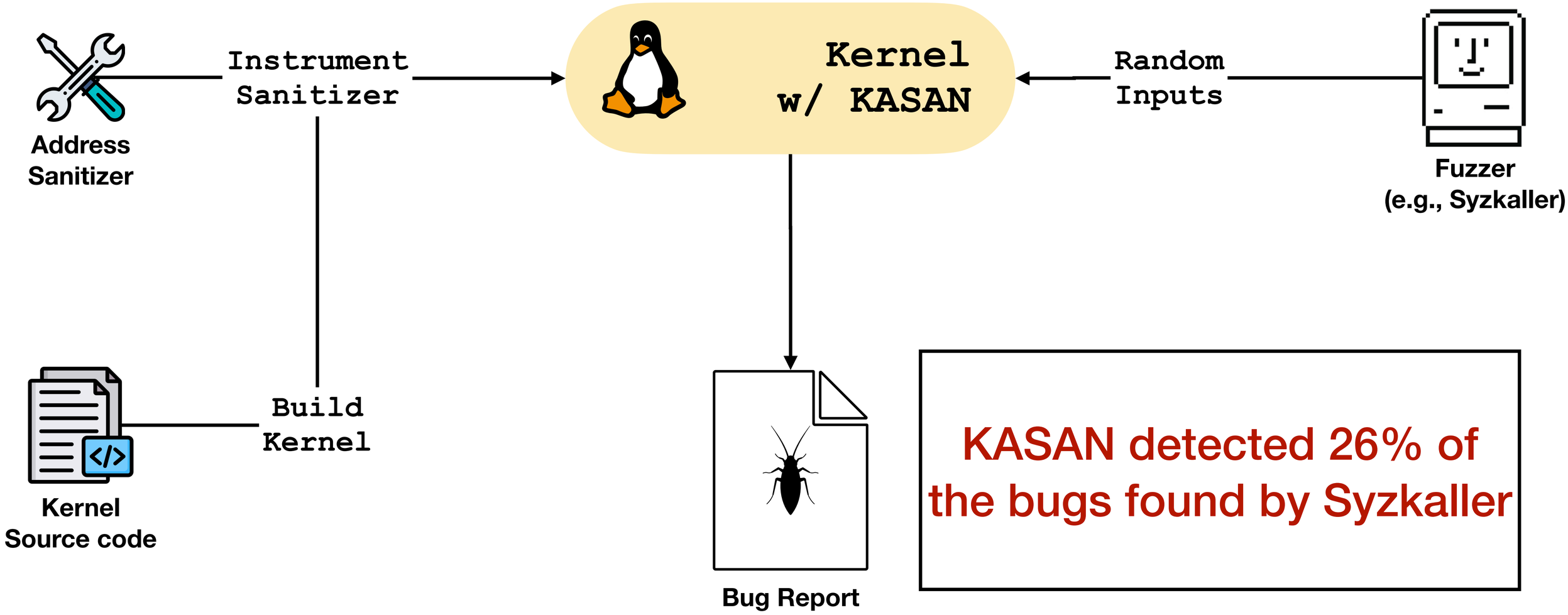
Apple investigating report of a new iOS exploit being used in the wild

Cyber-security firm ZecOps said today it detected attacks against high-profile targets using a new iOS email exploit.

Experts Unveil Exploit for Recent Windows Vulnerability Under Active Exploitation

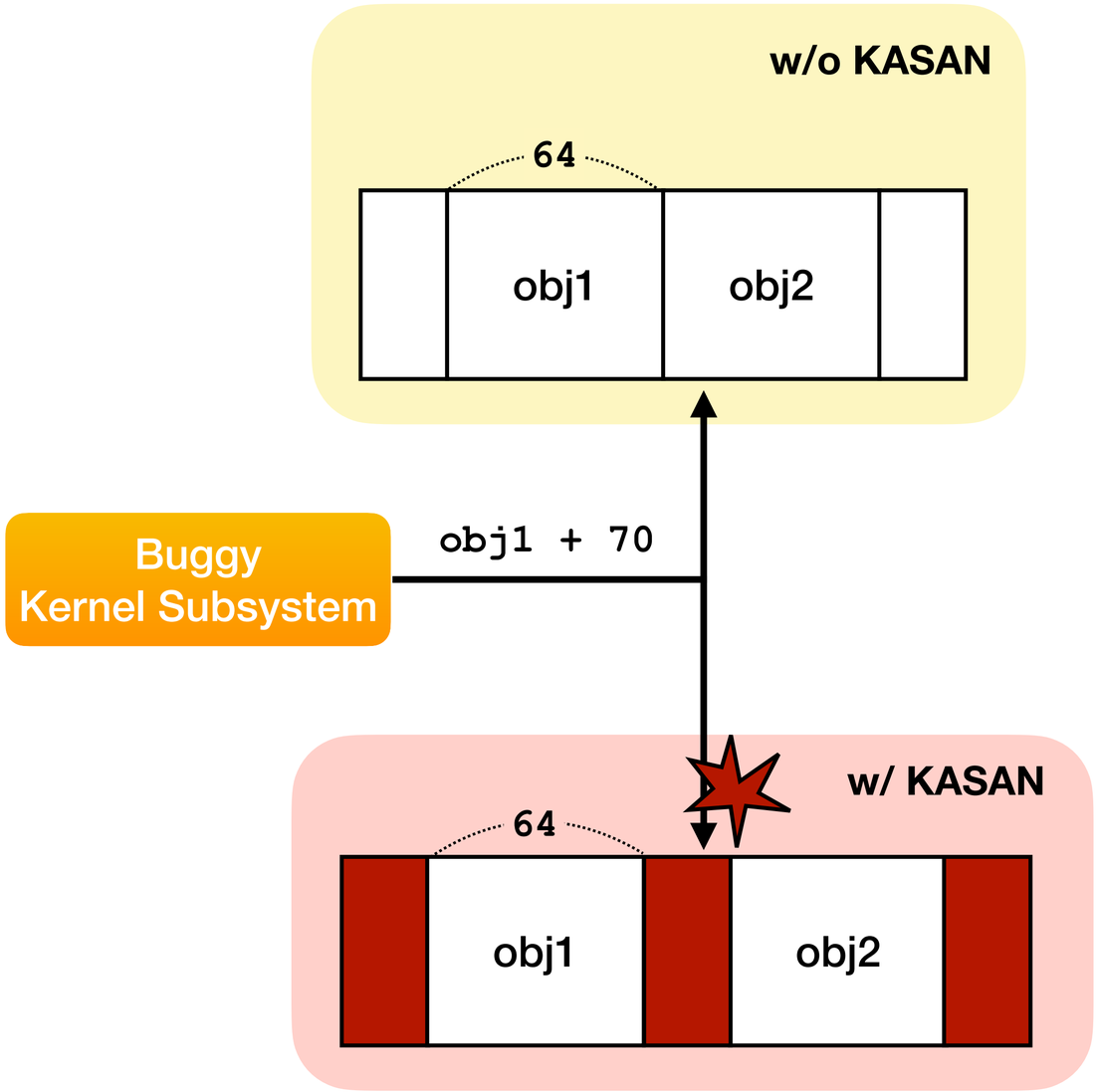
Jun 08, 2023 Ravie Lakshmanan Endpoint Security / Zero-Day

Kernel Fuzzer + KASAN



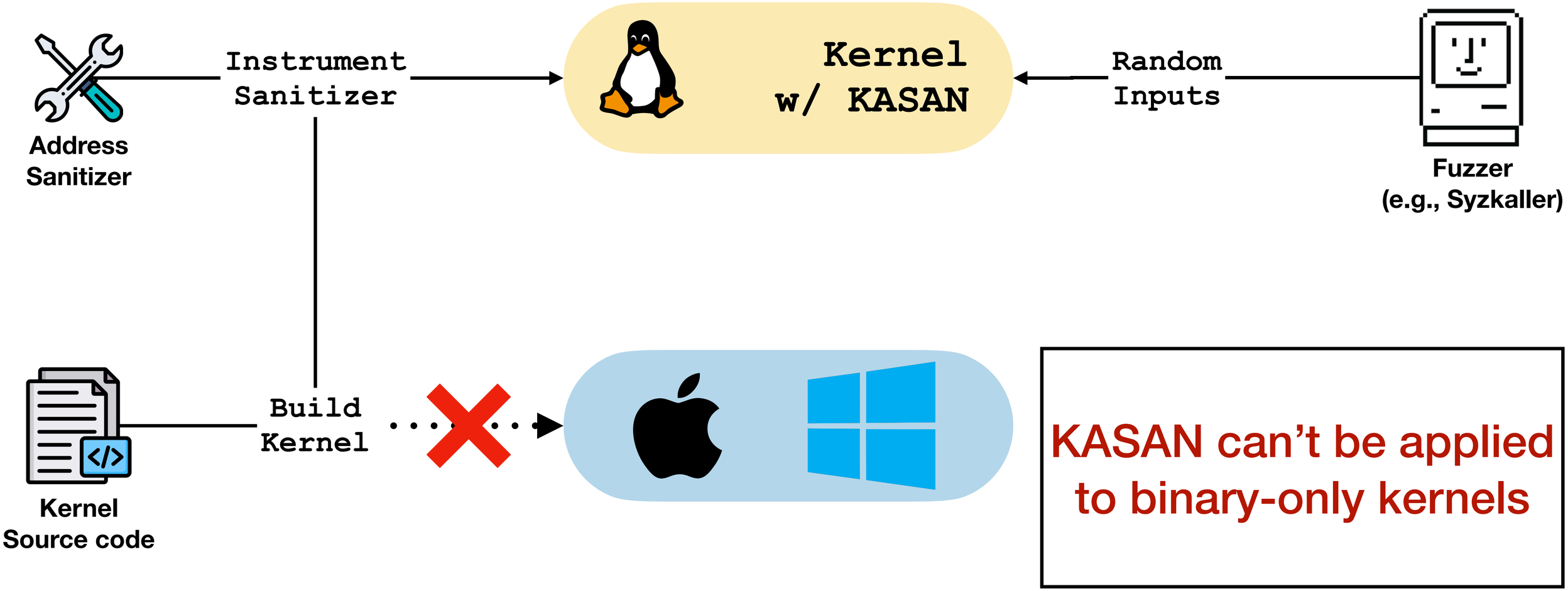
KASAN detected 26% of the bugs found by Syzkaller

Kernel Address Sanitizer



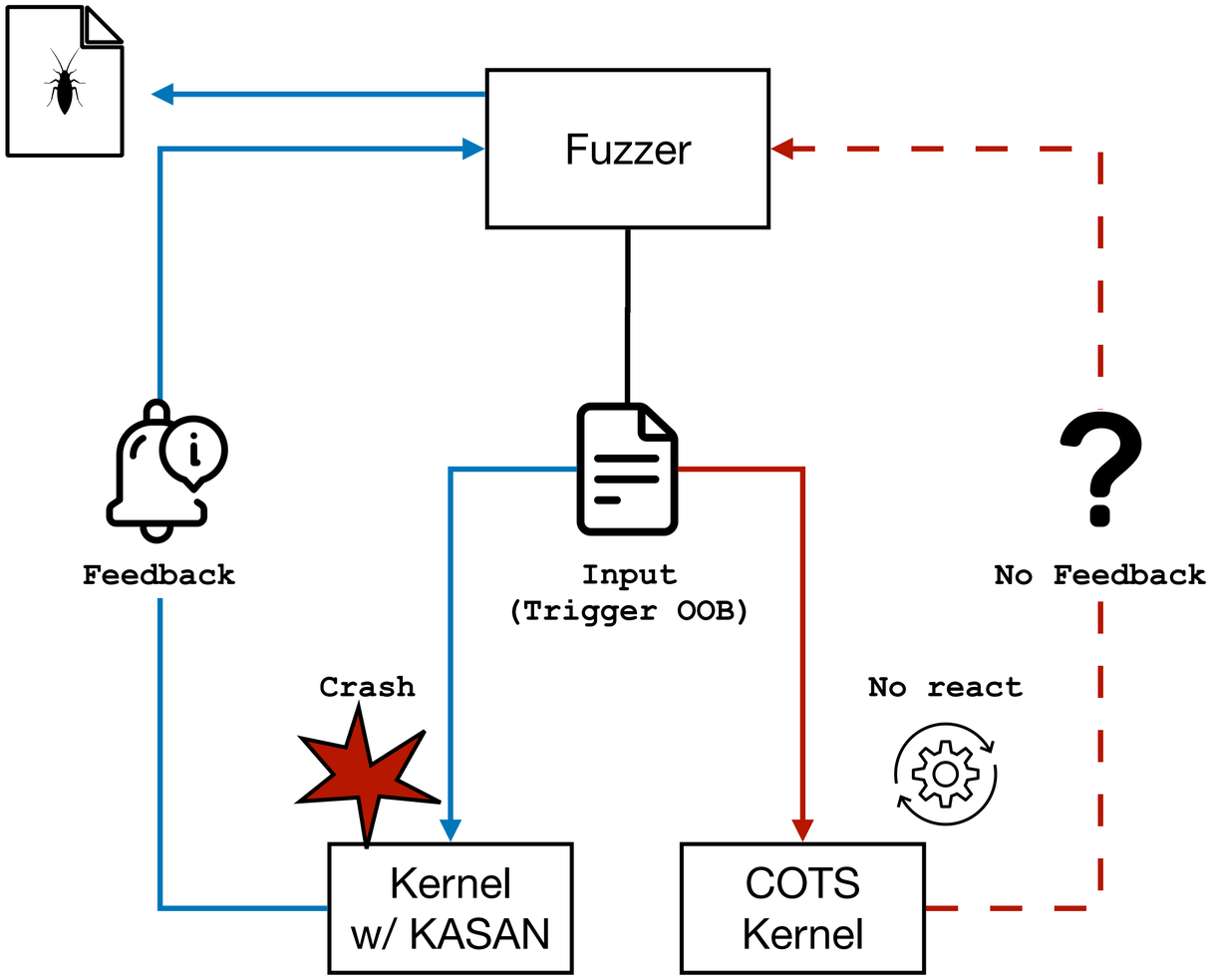
- **Dynamic memory error detector**
  - Using **source level** instrumentation
- **KASAN detects non-crashing bugs**
  - Via redzone-based detection
  - Use-After-Free, Out-of-Bounds Access

Binary-only Kernels?



KASAN can’t be applied to binary-only kernels

Challenge: KASAN + Binary OS

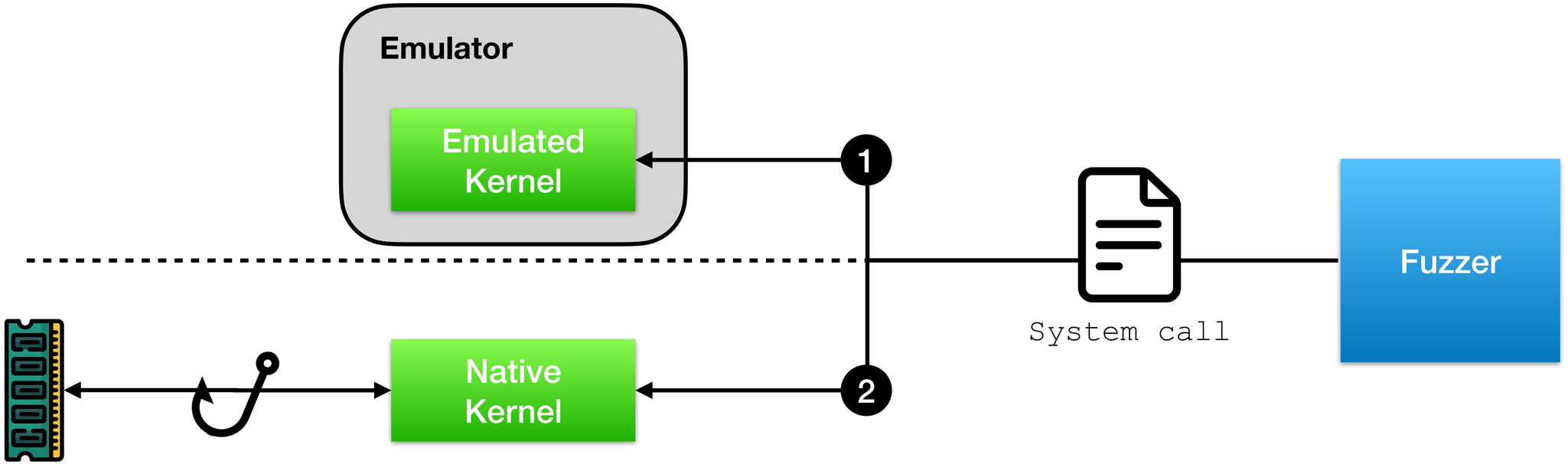


- **COTS OS vulnerabilities are critical**
  - Affect the most users
- **Most COTS OS are binary-only**
  - Like Windows, macOS
- **KASAN needs OS source code**
  - for **source-level instrumentation**
- **Binary-only approach is needed** for COTS OS

Way to binary-only: Static Instrumentation

- **Static instrumentation**
  - Perform static instrumentation such as **binary rewriting**
  - **Can achieve performance close to native execution**
  - **Difficult to ensure soundness**

Way to binary-only: Dynamic Instrumentation

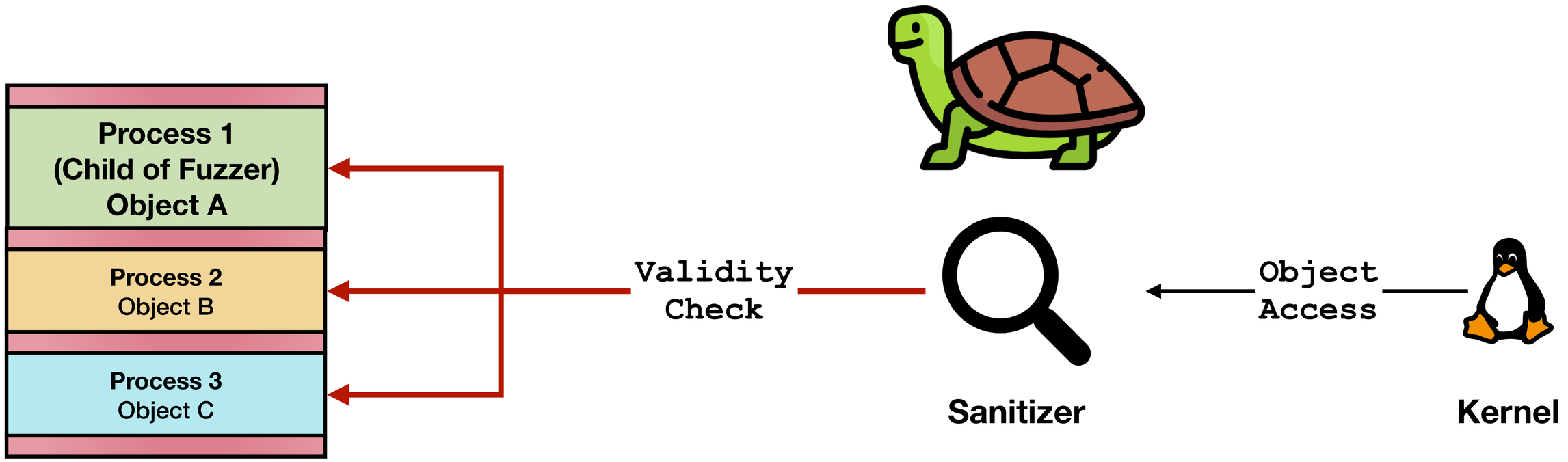


- **Dynamic Instrumentation**
  - **❶ Emulating the kernel** or **❷ Hooking a page fault handler**
- **Easy to Implement**
- **High performance overhead**
  - Full Emulation: ~85 times
  - Hook Page Fault Handler: ~644 times

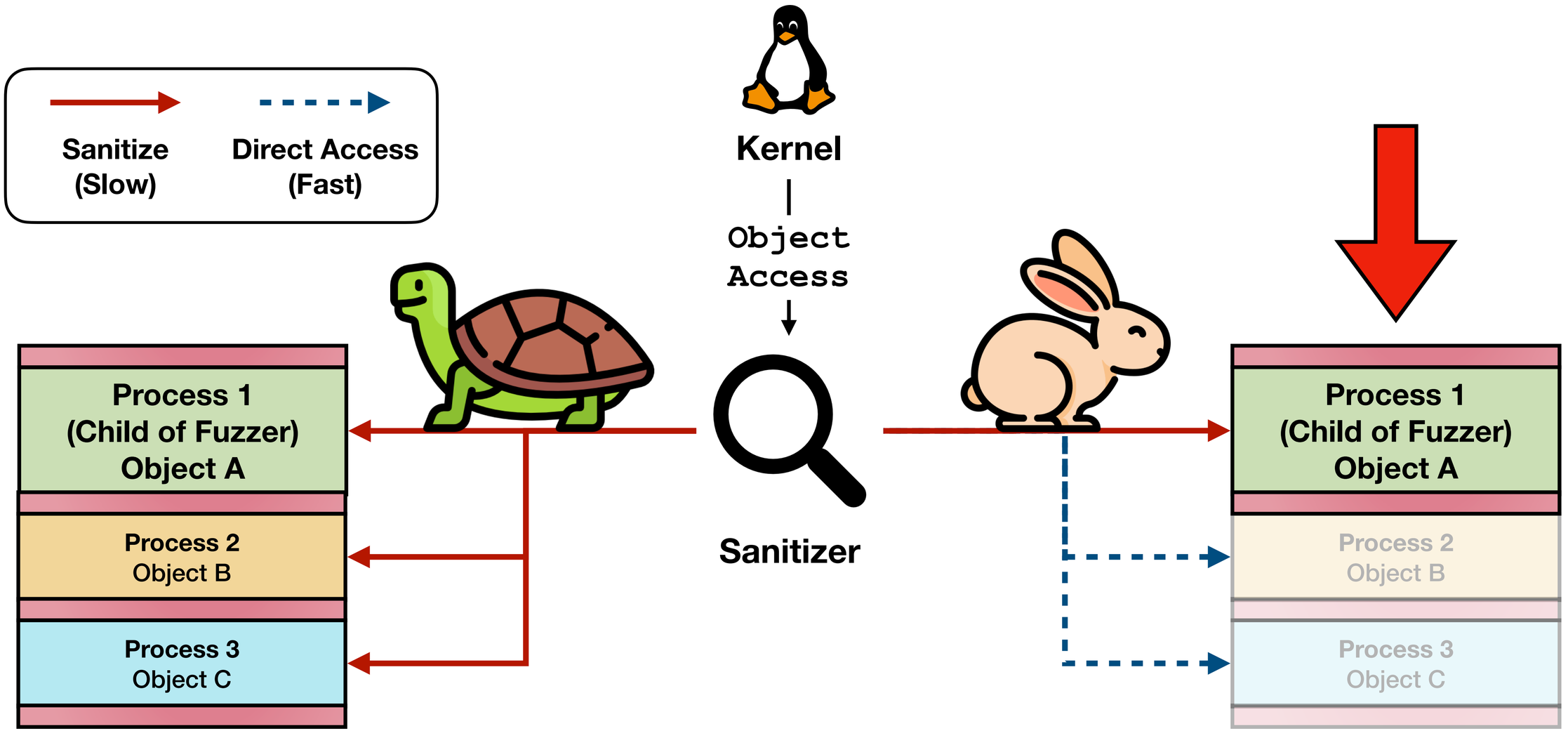
New Observation: Most crashes occur in fuzzer processes

	Triggered by	
Bug Type	**Fuzzer Process**	**Other**
UAF	**15**	1
OOB	**5**	1
Total	**20**	2

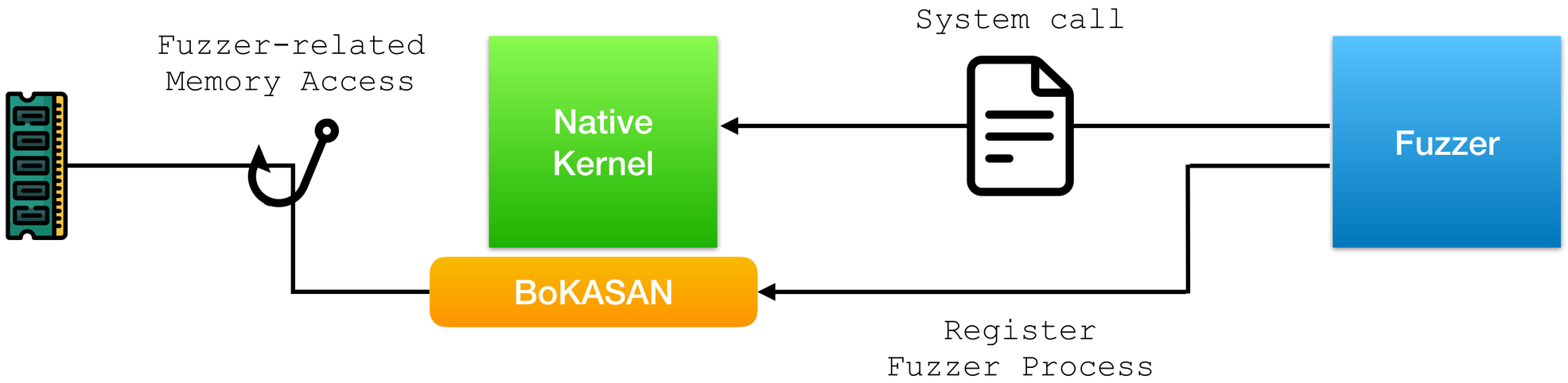
Sanitize all memory objects causes overhead



Our Insight: Focus Solely on the target

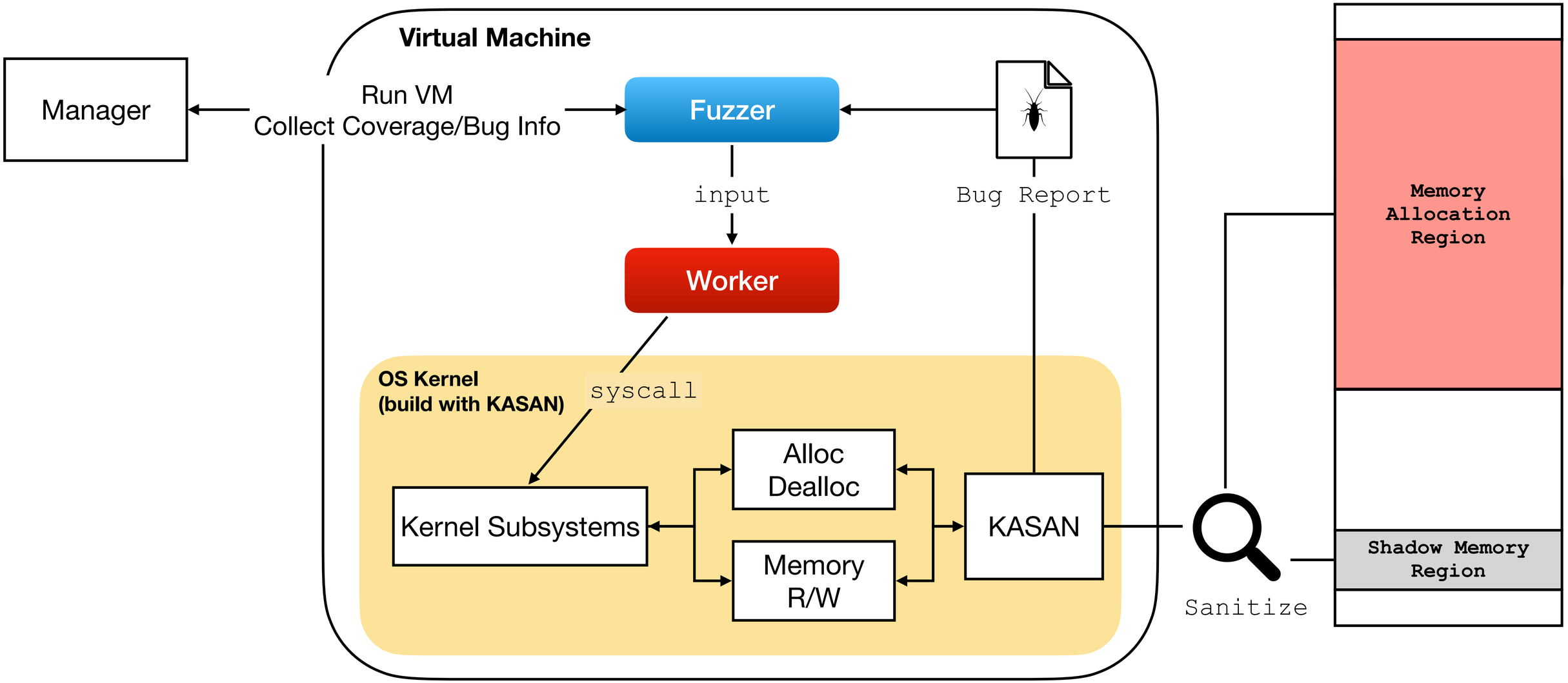


Basic Concept: Selective Sanitization

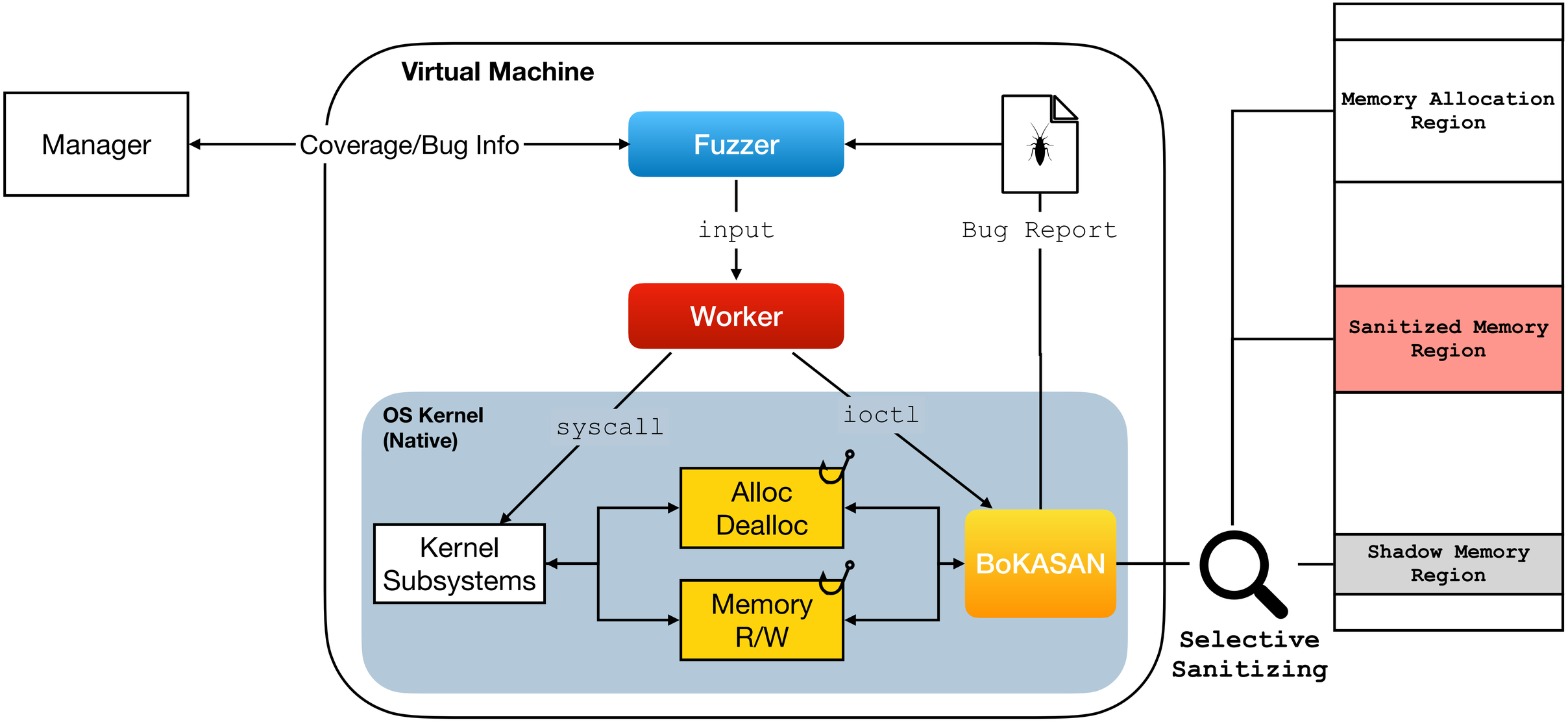


- **Page fault-based Selective Sanitization**
  - Sanitize **fuzzer-generated inputs only**
    - Minimized page fault overhead
    - Possible in all COTS OS via paging
- We introduce **BoKASAN, the first binary-only KASAN**

Typical Kernel Fuzzing Flow

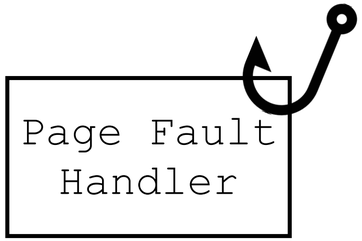


BoKASAN: Design Overview



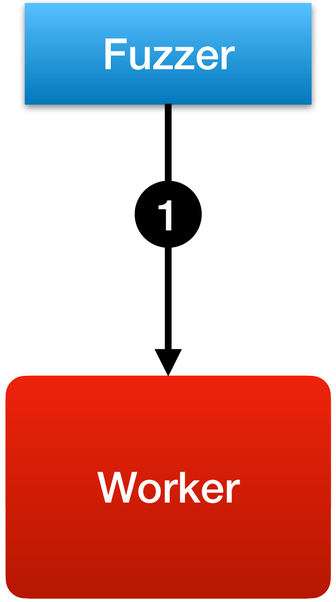
Fuzzing workflow of BoKASAN

BoKASAN



0. At boot time, BoKASAN hooks kernel memory functions and fault handlers

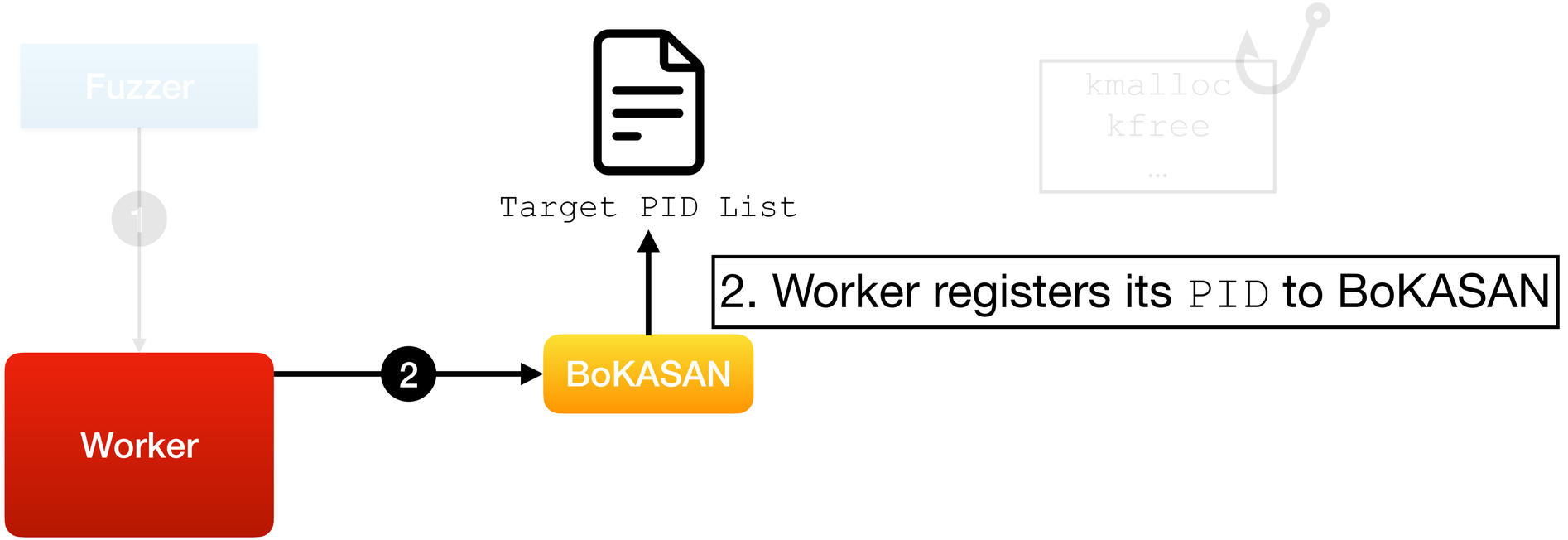
Fuzzing workflow of BoKASAN



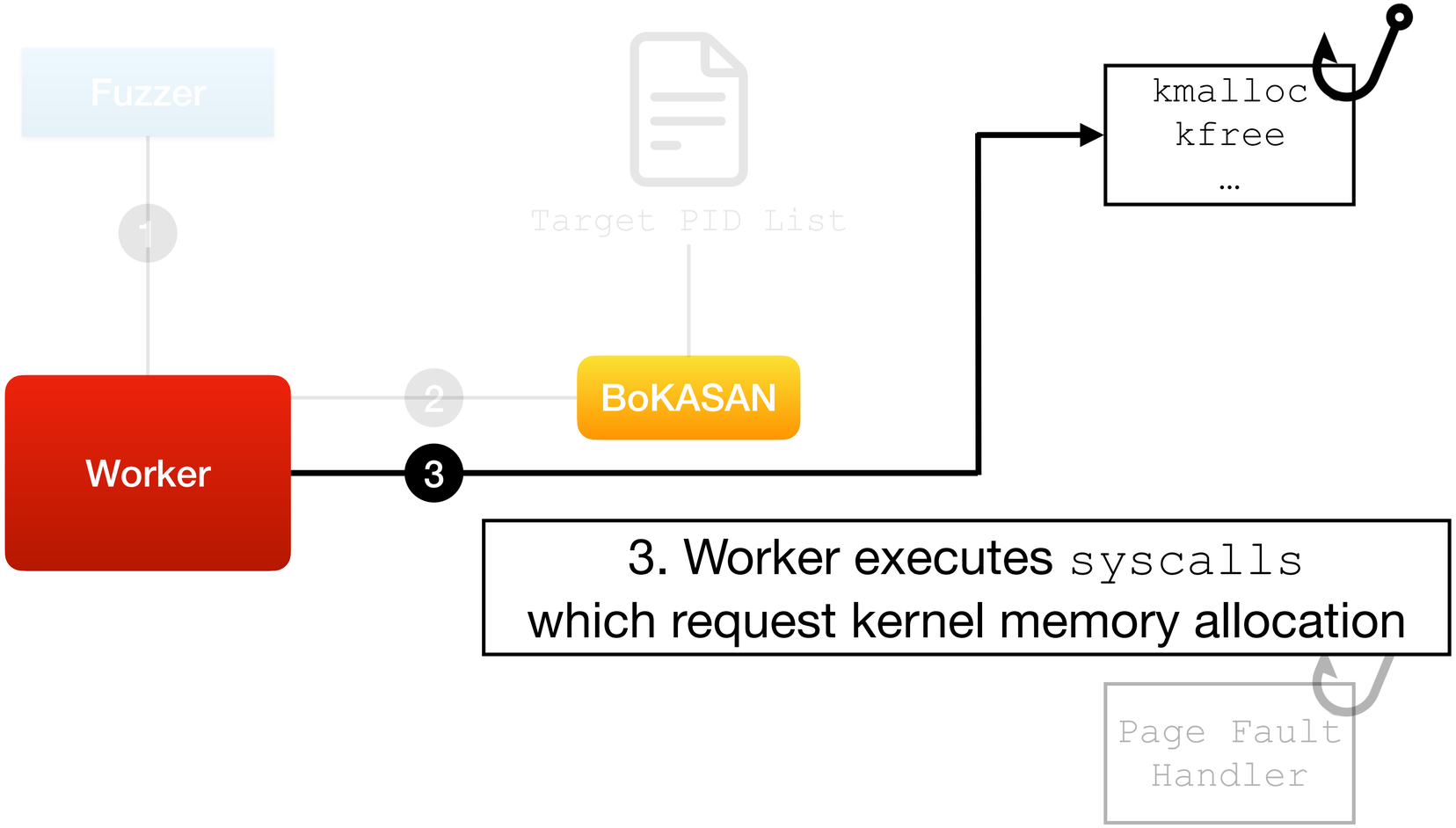
1. Fuzzer spawns a worker to execute `syscall` it generates.

BoKASAN

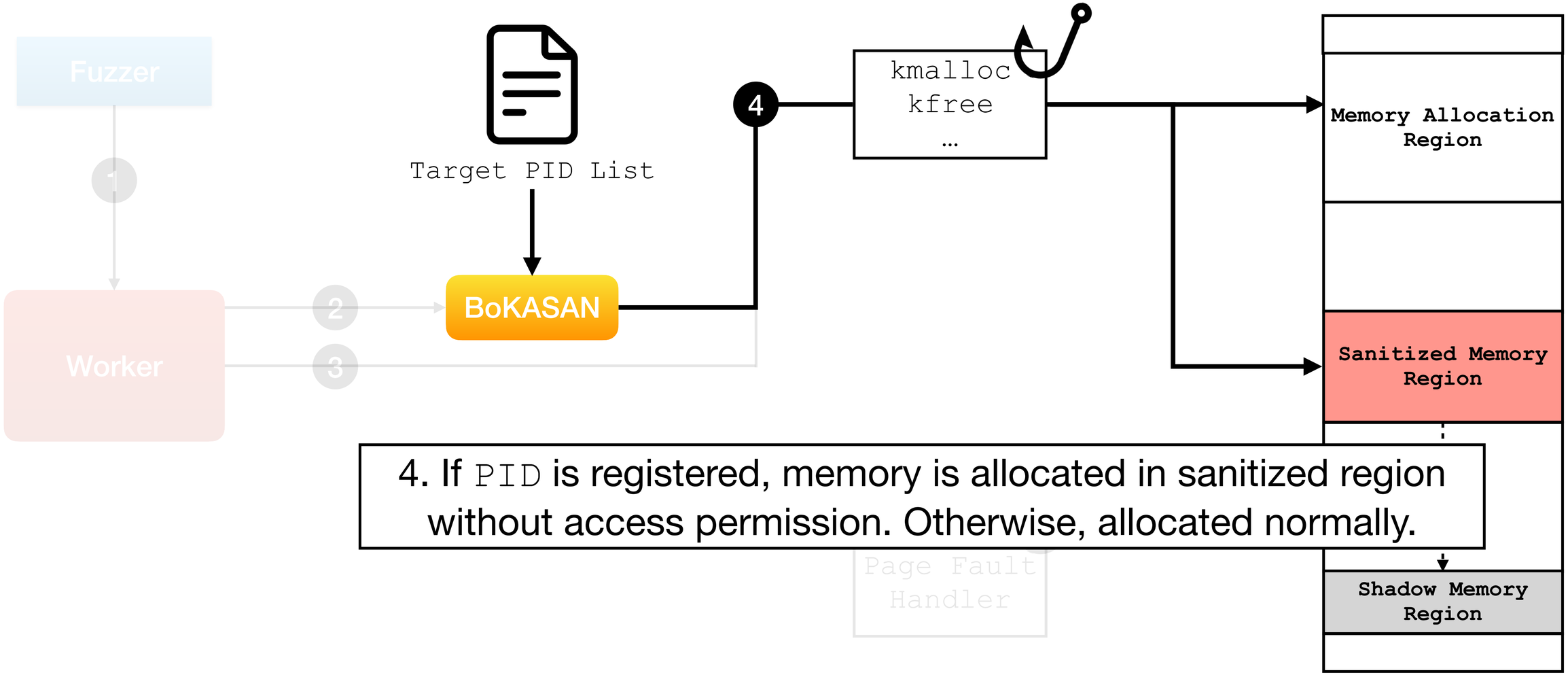
Fuzzing workflow of BoKASAN



Fuzzing workflow of BoKASAN

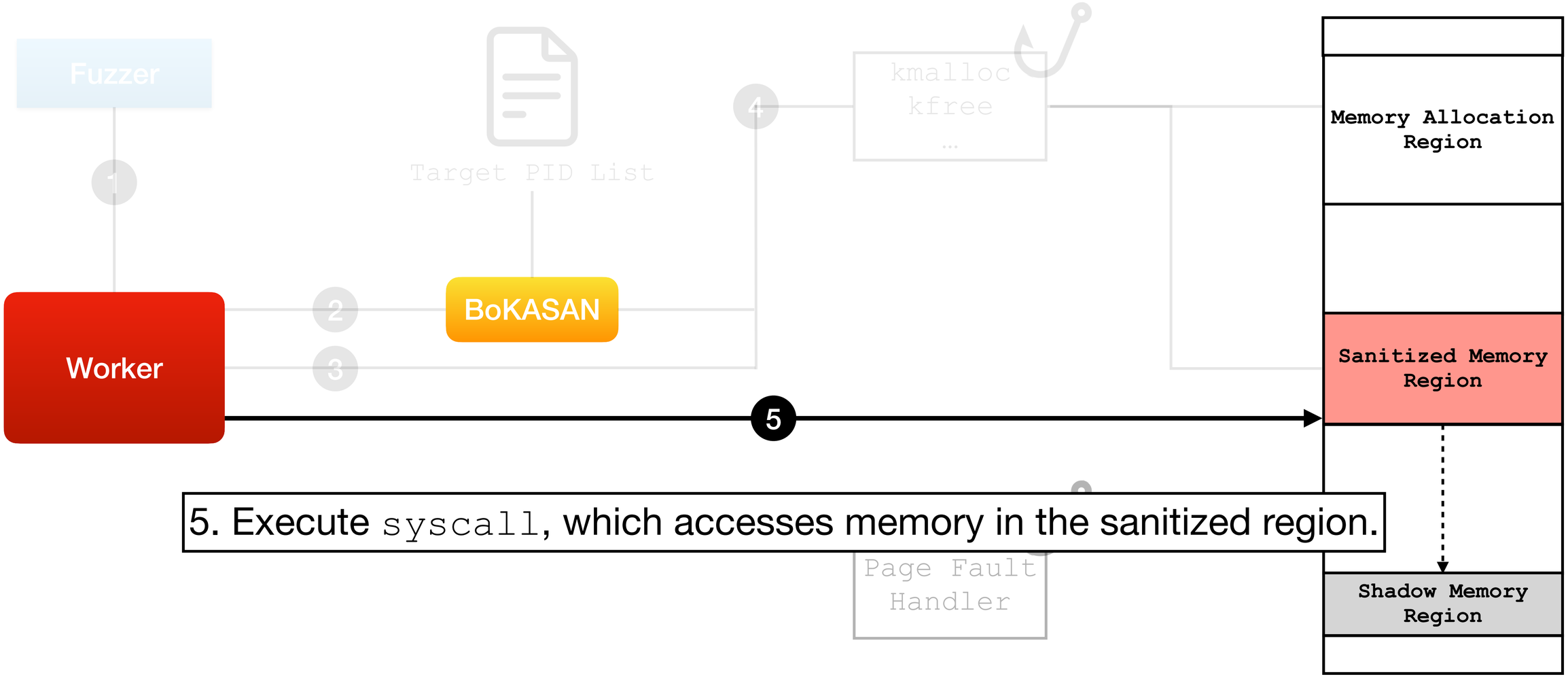


Fuzzing workflow of BoKASAN



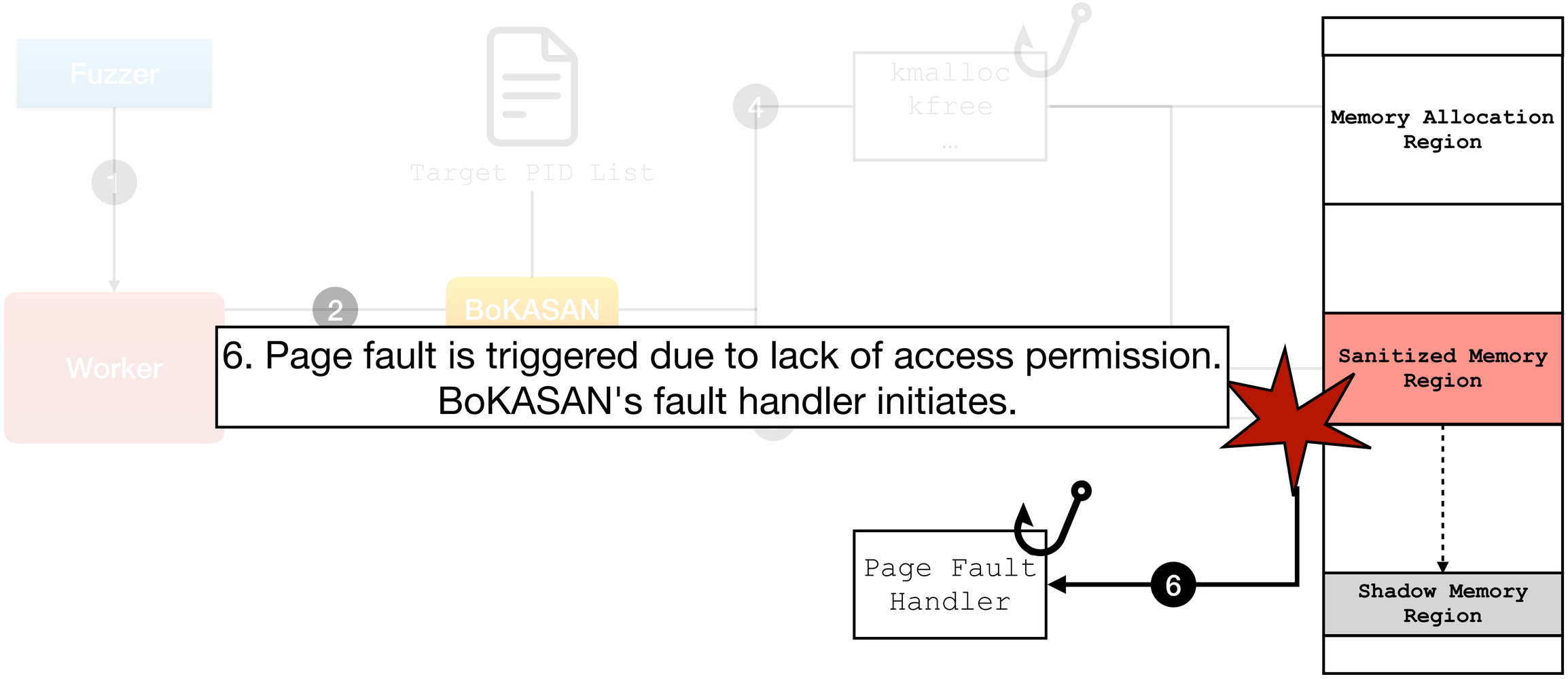
4. If `PID` is registered, memory is allocated in sanitized region without access permission. Otherwise, allocated normally.

Fuzzing workflow of BoKASAN



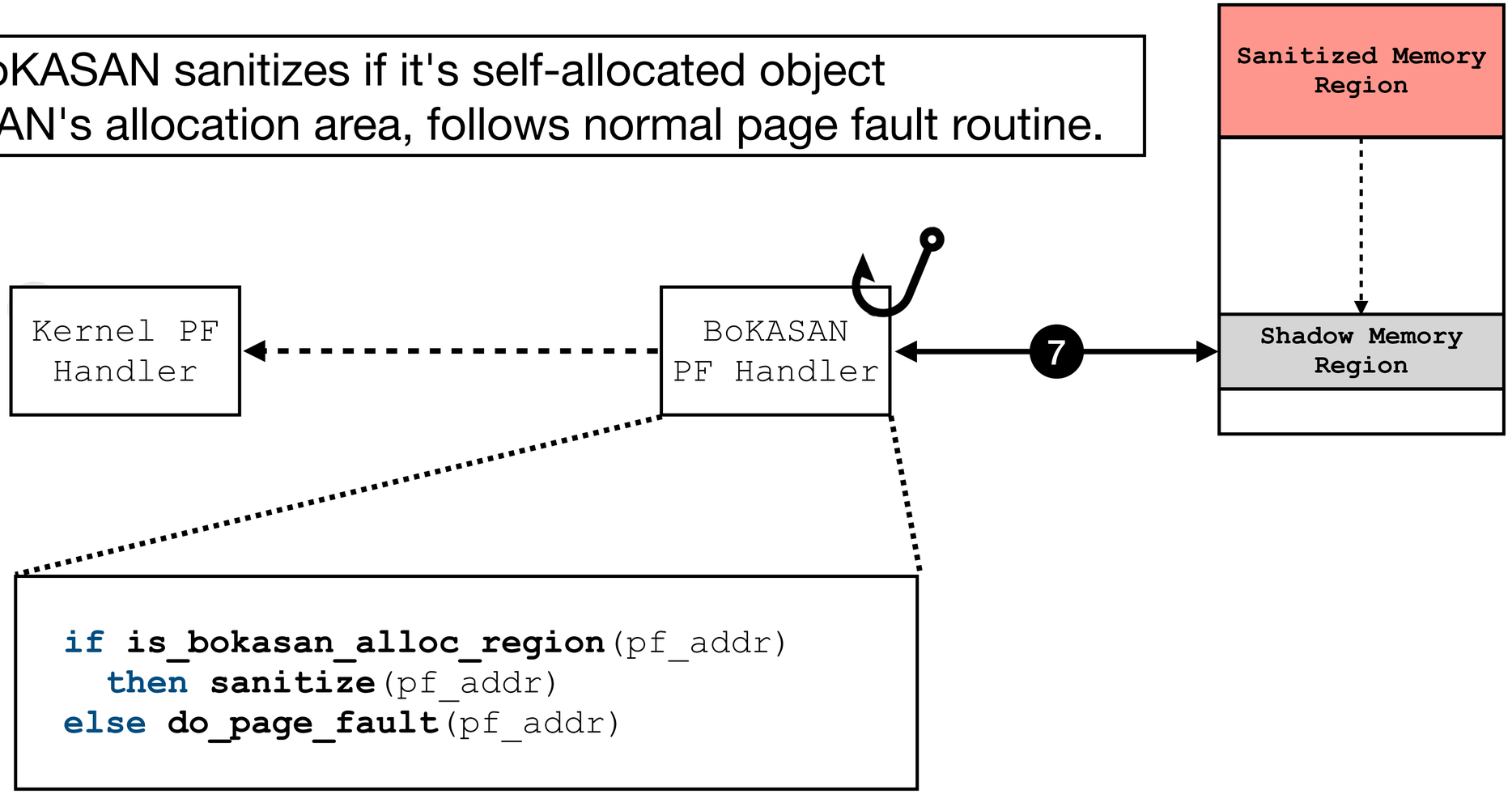
5. Execute `syscall`, which accesses memory in the sanitized region.

Fuzzing workflow of BoKASAN

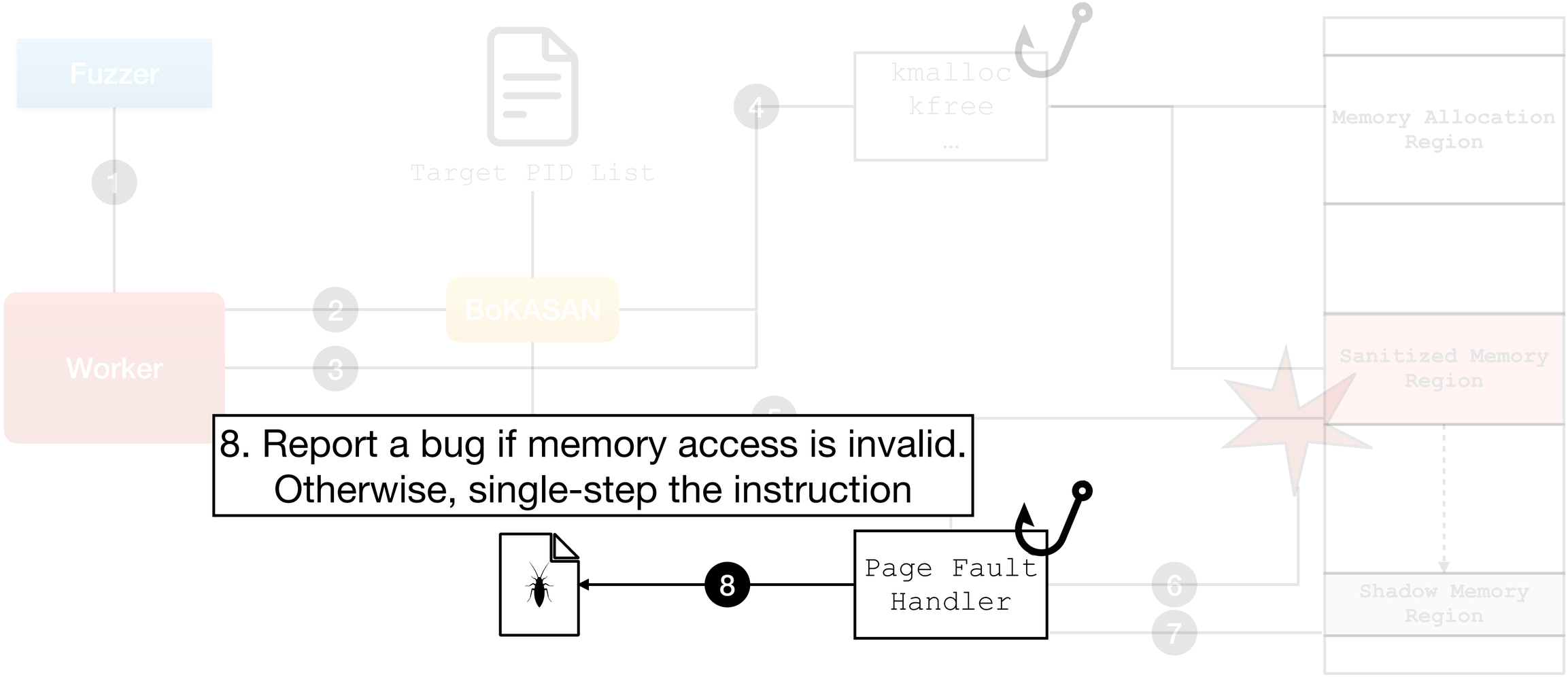


Fuzzing workflow of BoKASAN

7. BoKASAN sanitizes if it's self-allocated object
If not in BoKASAN's allocation area, follows normal page fault routine.



Fuzzing workflow of BoKASAN



Evaluation: Setup

- **Dataset**
  - Tested with bugs found in
    - SyzVegas ’21 Security
    - Janus ’19 S&P
  - 23 OOB & UAF Bugs
- **Sanitizers**
  - KASAN (Native)
  - KASAN (Fully Emulated with QEMU)
- **Platform and Configuration**
  - Ubuntu 16.04
    - Intel Xeon Gold 6148, 384GB RAM
  - Ubuntu 20.04
    - Intel i7-12700, 64GB of RAM
- **Fuzzers**
  - Syzkaller
    - commit #fdb2bb2c23ee7

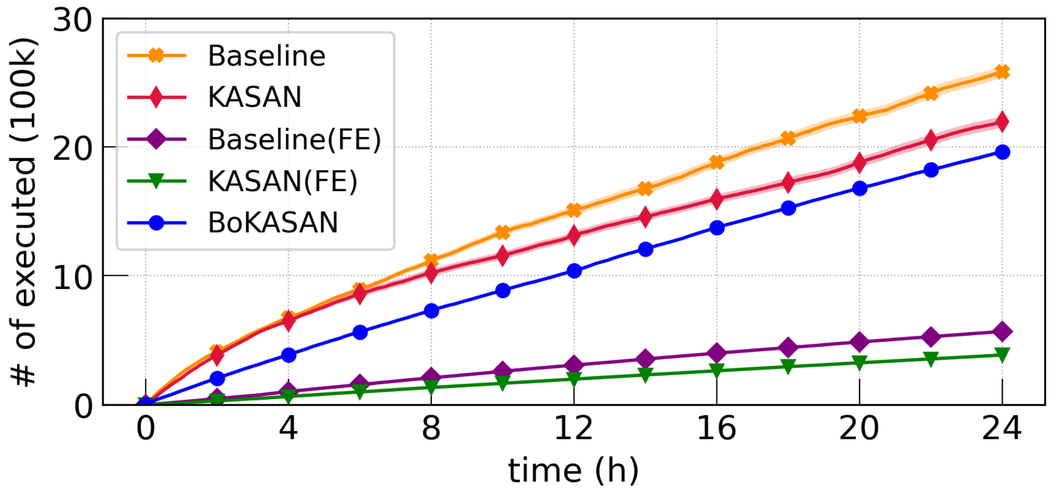
Evaluation: Bug Detection

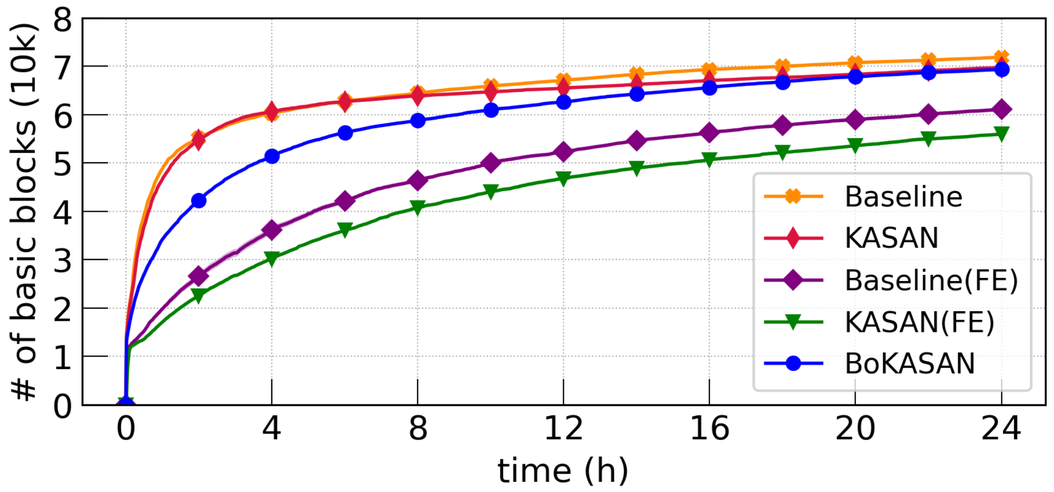
- **BoKASAN** detected a similar number of bugs to KASAN
  - Janus dataset: 20 vs. **21**
  - SyzVegas dataset: 15 vs. **14**

#	Function	Version	Type	KA.	BoK.
1	vcs_write	4.19	OOB	✓	✓
2	ata_scsi_mode_select_xlat	4.19	UAF	✓	✓
3	clear_buffer_attributes	4.19	UAF	✓	✓
4	complement_pos	4.19	UAF	✓	✓
5	con_scroll	4.19	UAF	✓	✓
6	con_shutdown	4.19	UAF	✓	✓
7	do_con_write	4.19	UAF	✓	✓
8	do_update_region	4.19	UAF	✓	✓
9	get_work_pool_id	4.19	UAF	✓	✗
10	screen_glyph_unicode	4.19	UAF	✓	✓
11	vc_do_resize	4.19	UAF	✓	✓
12	vcs_write	4.19	UAF	✓	✓
13	vc_uniscr_check	4.19	UAF	✓	✓
14	vgacon_invert_region	4.19	UAF	✓	✓
15	vgacon_scroll	4.19	UAF	✓	✓
Total				15	14

#	FS	Report ID	Version	Type	KASAN	BoKASAN
1	EXT4	199181	4.15	OOB	✗	✗
2		199347	4.16-rc1	OOB[1]	✓	✓
3		199403	4.16-rc1	UAF	✓	✓[2]
4		199417	4.16-rc1	OOB	✓	✓
5		199865	4.17-rc4	OOB	✓[2]	✓
6		200001	4.17-rc4	OOB	✓[3]	✓[2]
7		200401	4.17-rc4	OOB[1]	✓	✓[2]
8		200931	4.18	UAF	✓	✓
9	XFS	199371	4.16-rc1	UAF	✓	✓
10		199373	4.16-rc1	UAF	✓	✓
11		199381	4.15.13	OOB[1]	✗	✓
12		199443	4.17-rc1	UAF	✓	✓[2]
13		200047	4.17-rc4	OOB	✓	✓
14		200053	4.17-rc4	OOB	✓	✓
15		200923	4.18	OOB	✓	✓
16	BTRFS	199837	4.17-rc5	OOB	✗	✗
17		199839	4.17-rc5	UAF	✓	✓[2]
18	F2FS	200167	4.18-rc1	OOB	✓[3]	✓[2]
19		200173	4.18-rc1	OOB	✓	✓
20		200179	4.18-rc1	UAF	✓	✓[2]
21		200219	4.18-rc1	OOB[1]	✓	✓
22		200419	4.16-rc1	OOB	✓	✓
23		200421	4.18-rc3	OOB	✓	✓[2]
Total					20	21

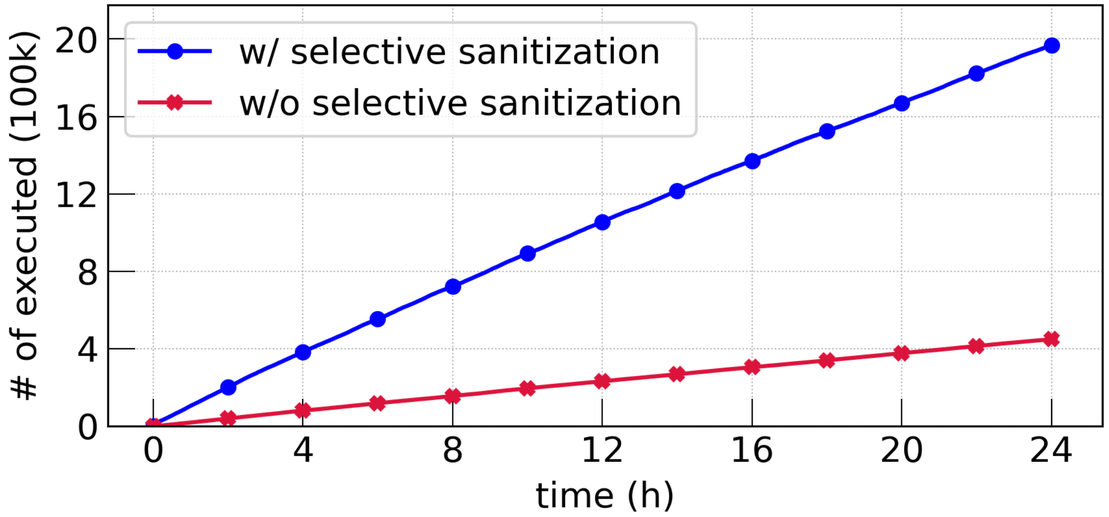
Evaluation: Performance Overhead

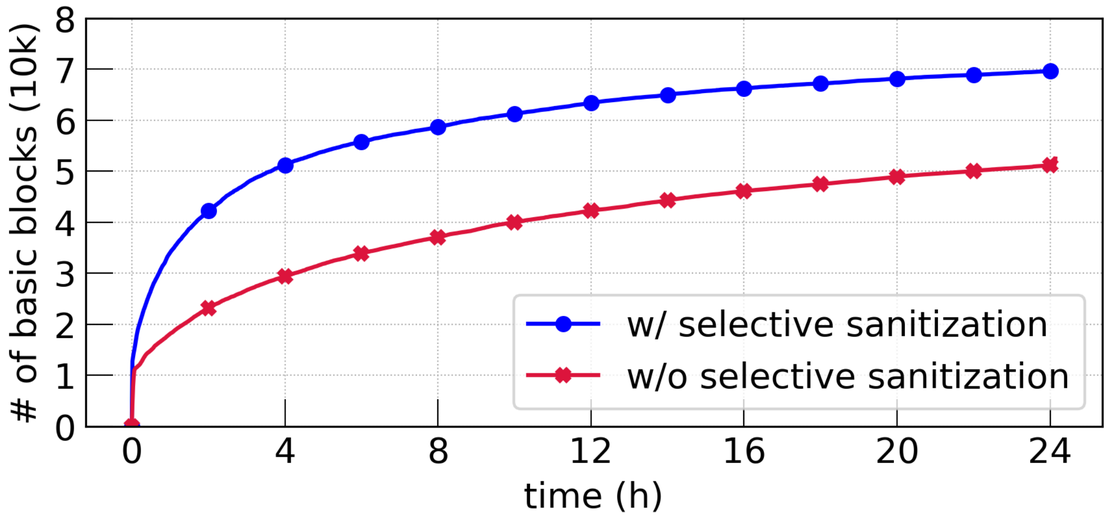




- **24h Fuzzing Experiment**
  - **Executed Syscalls**
    - 81.3% more than KASAN(FE)
    - 11.3% less than KASAN
  - **Covered Basic Blocks**
    - 12.0% more than KASAN(FE)
    - 0.6% less than KASAN
- BoKASAN's performance is
  - **Significantly better** than emulated KASAN
  - **Almost similar** to KASAN

Evaluation: Selective Sanitization





- **Fuzzing with Selective Sanitization**
  - Executed **4.3x** more `syscall`
  - Covered **32.8%** more basic blocks
  - Found **8** more bugs
- **Shows the effectiveness of selective sanitization**

Evaluation: Binary-only Kernel Fuzzing

- **Binary-only fuzzing with kAFL**
  - kAFL's `kafl_vuln_test` driver
- **Target kernel**
  - Windows 10 21H2, Ubuntu 16.04
- **The driver contains three bugs**
  - Two bugs can be found without KASAN (❶, ❷)
  - One bug can only be found with KASAN (❸)

```
13      if (userBuffer[0] == 'K')
14       if (userBuffer[1] == 'E')
15        if (userBuffer[2] == 'R')
16         if (userBuffer[3] == 'N')
17          if (userBuffer[4] == 'E')
18           if (userBuffer[5] == 'L')
19            if (userBuffer[6] == 'A')
20             if (userBuffer[7] == 'F')
21    ❶         if (userBuffer[8] == 'L')
22                  ((VOID(*)())0x0)();
23      if (userBuffer[0] == 'S')
24       if (userBuffer[1] == 'E')
25        if (userBuffer[2] == 'R')
26         if (userBuffer[3] == 'G')
27          if (userBuffer[4] == 'E')
28           if (userBuffer[5] == 'J')
29    ❷         size = *((PSIZE_T)(0x0));
```

```
30      if (userBuffer[0] == 'K')
31       if (userBuffer[1] == 'A')
32        if (userBuffer[2] == 'S')
33         if (userBuffer[3] == 'A')
34          if (userBuffer[4] == 'N') {
35            ExFreePool(array);
36    ❸       array[0] = 1234;
37            return STATUS_SUCCESS;
38          }
```

Evaluation: Binary-only Kernel Fuzzing

- **Bug case ❶, ❷**
  - Bug is detected in all trials w/ and w/o BoKASAN
- **Bug case ❸**
  - With BoKASAN
    - Detected 20 times on both OSes
  - Without BoKASAN
    - Detected 3 times on Ubuntu
    - No detection on Windows
- **BoKASAN is applicable to binary-only kernels**

```
30      if (userBuffer[0] == 'K')
31       if (userBuffer[1] == 'A')
32        if (userBuffer[2] == 'S')
33         if (userBuffer[3] == 'A')
34          if (userBuffer[4] == 'N') {
35            ExFreePool(array);
36        3   array[0] = 1234;
37            return STATUS_SUCCESS;
38          }
```

Target	Sanitizer	Time to Crash (s) Bug1	Bug2	Bug3	#Execs
Linux	Baseline	8.72 (20)	3.88 (20)	1.07 (3)	1,482,702
	BoKASAN	20.56 (20)	4.87 (20)	6.68 (20)	997,259
Windows	Baseline	48.63 (20)	25.32 (20)	- (0)	722,437
	BoKASAN	54.23 (20)	28.71 (20)	75.50 (20)	288,771

Conclusion

- **BoKASAN: first practical binary-only KASAN which can be used for COTS OS**
- Reduces binary instrumentation overhead with **selective sanitization**
- BoKASAN is anticipated to aid future research in binary-only kernel fuzzing

YONSEI UNIVERSITY
INFORMATION SECURITY &
AI SECUTIY LABORATORY

Thank you!

1st Author: Mingi Cho (imgc@yonsei.ac.kr)

Presentator: Dohyeon An (overflow@yonsei.ac.kr)

Corr. Author: Taekyoung Kwon (taekyoung@yonsei.ac.kr)

https://github.com/seclab-yonsei/bokasan