PeriScope: An Effective Probing and Fuzzing Framework for the Hardware-OS Boundary

Dokyung Song, Felicitas Hetzelt, Dipanjan Das, Chad Spensky, Yeoul Na, Stijn Volckaert, Giovanni Vigna, Christopher Kruegel, Jean-Pierre Seifert, Michael Franz

UCI Technische Universität Berlin UCSB KU LEUVEN

Remote compromise of peripheral chips

ars TECHNICA

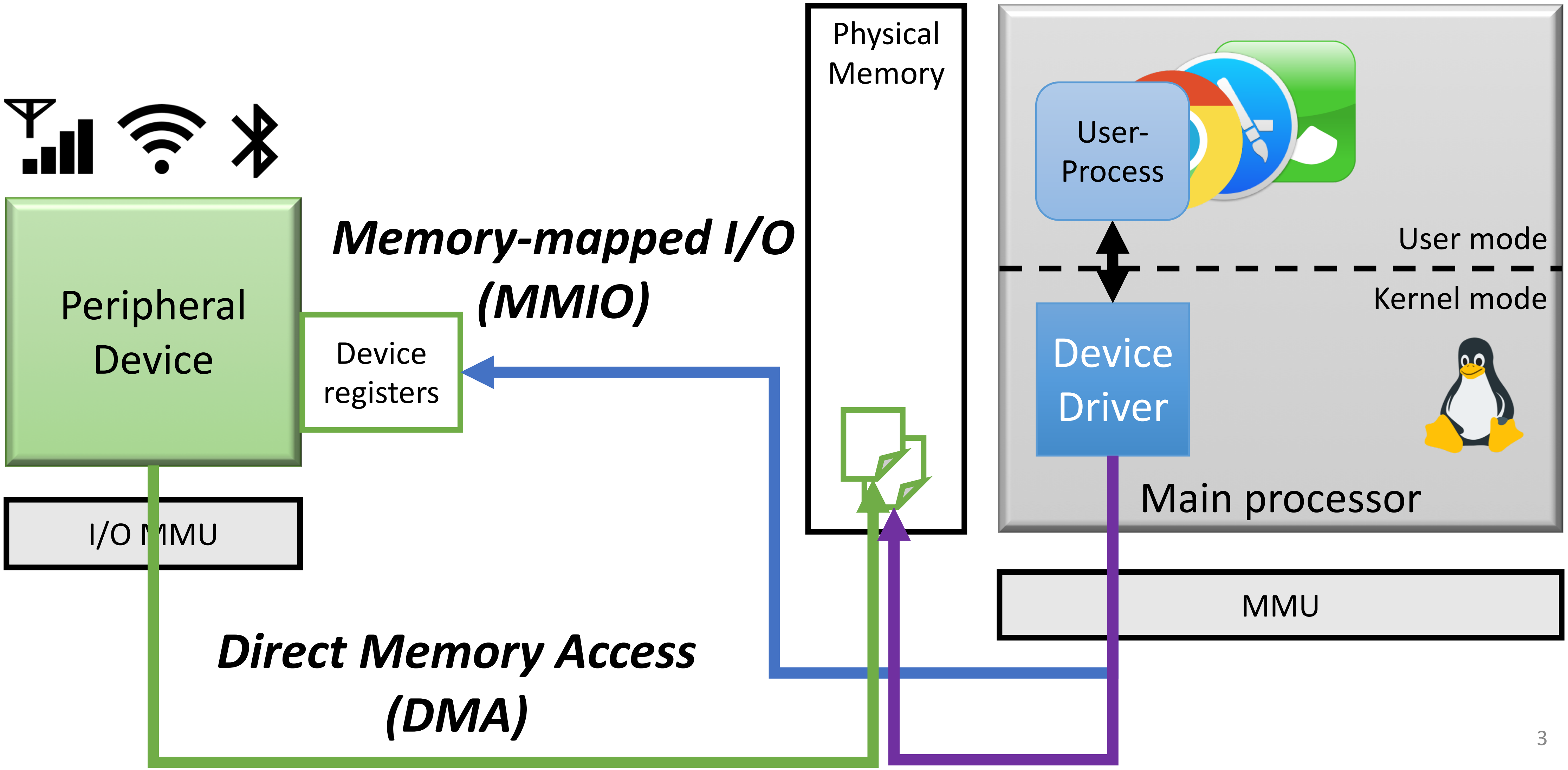
BIZ & IT —

Broadcom chip bug opened 1 billion phones to a Wi-Fi-hopping worm attack

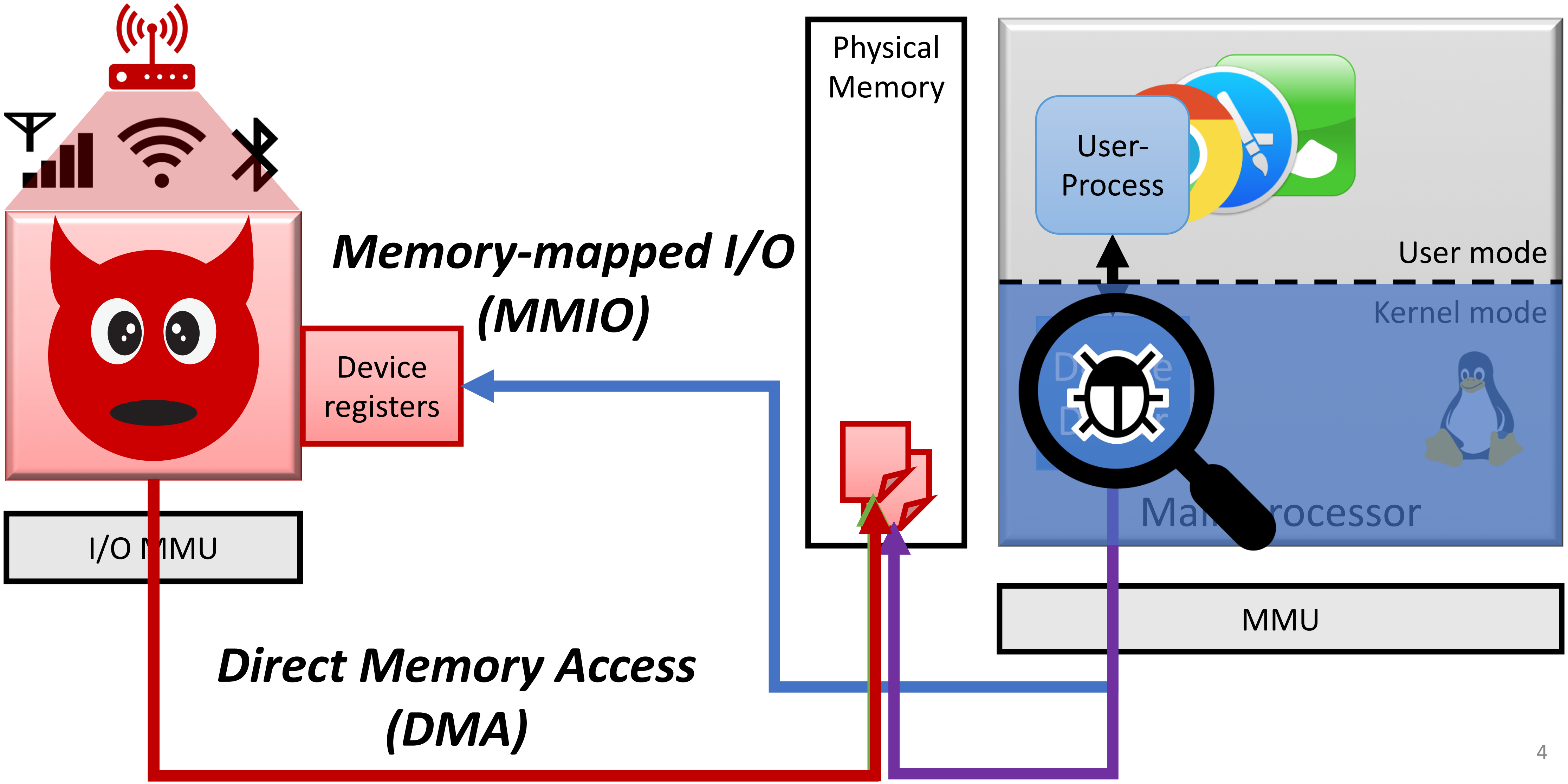
Wi-Fi chips used in iPhones and Android may revive worm attacks of old.

DAN GOODIN - 7/28/2017, 12:35 PM

Hardware-OS Interface: MMIO and DMA

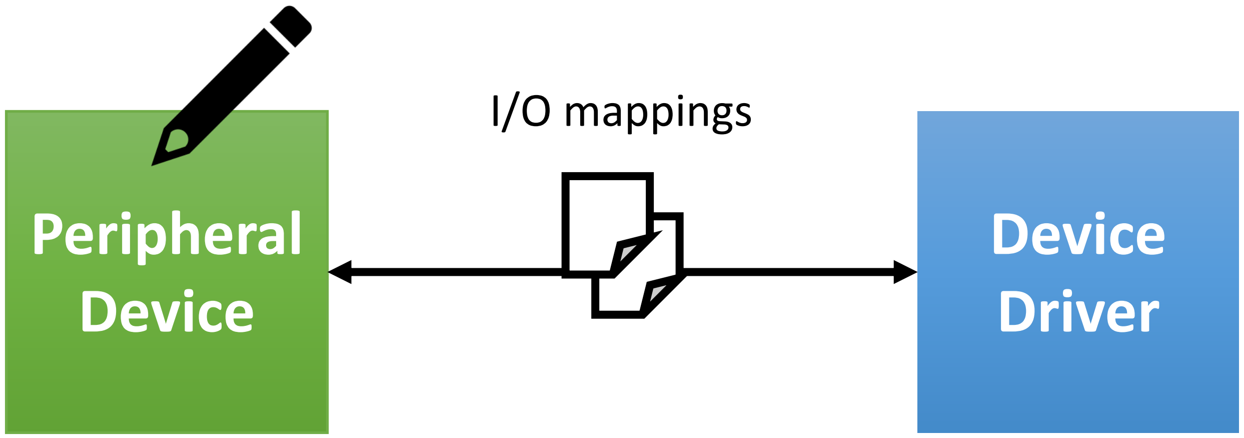


Threat Model



State-of-the-art: Analyzing HW-OS Interface (1/3)

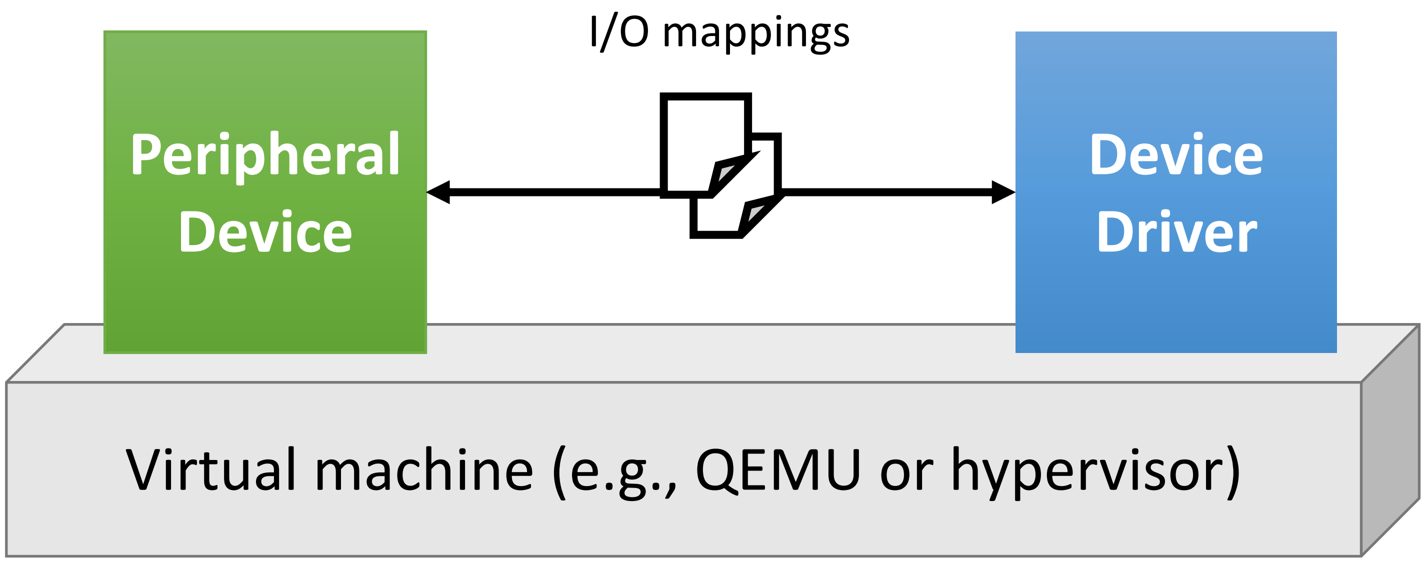
- ***Device Adaptation***
  - Pros: Non-intrusive (OS-independent)
  - Cons: Need for programmable device + limited visibility into driver



Reprogram the device
(e.g., FaceDancer21 custom USB)

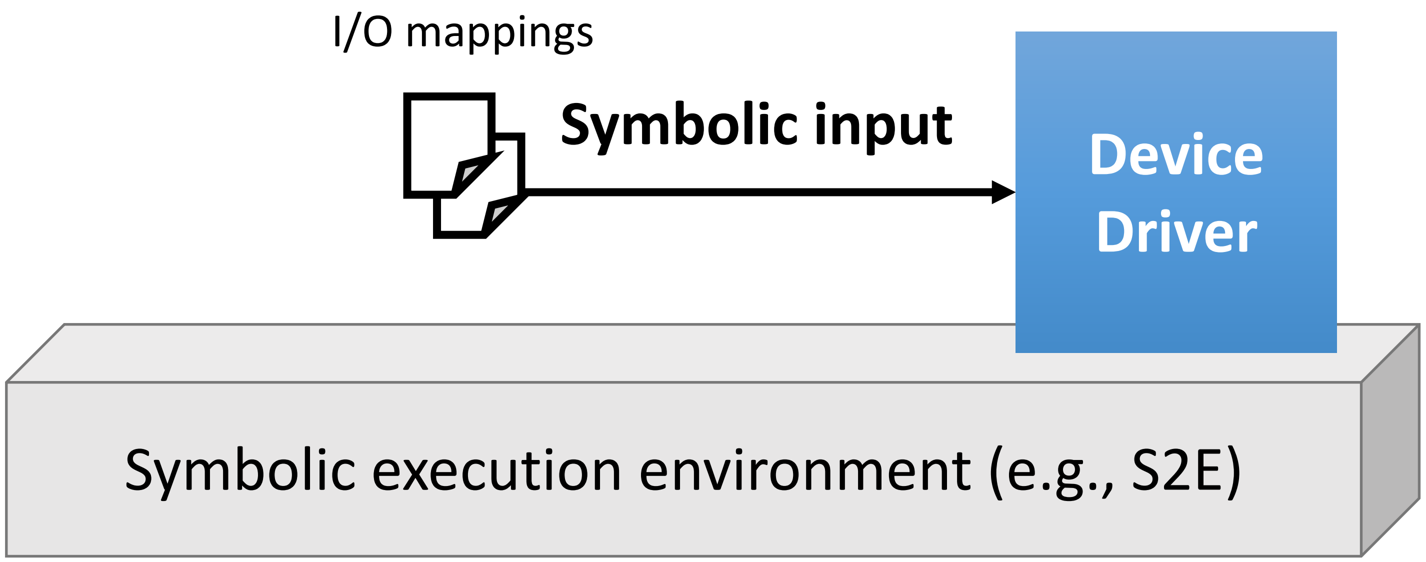
State-of-the-art: Analyzing HW-OS Interface (2/3)

- ***Virtual Machine***
  - Pros: High visibility yet non-intrusive
  - Cons: Need for virtual device and/or virtualization HW support



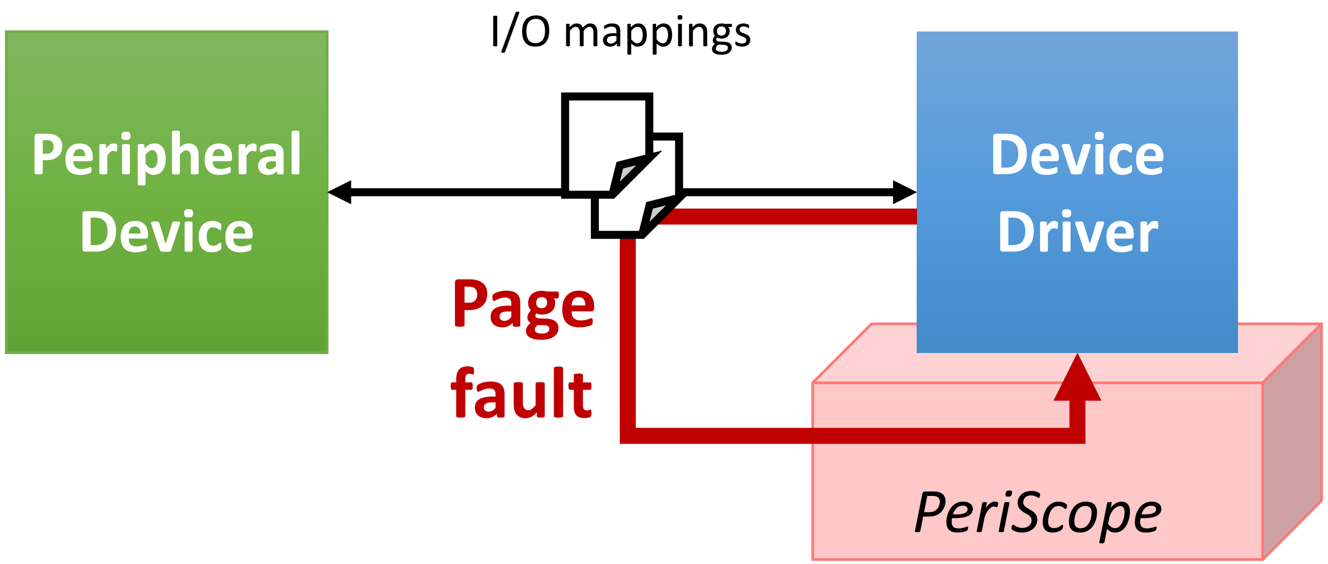
State-of-the-art: Analyzing HW-OS Interface (3/3)

- ***Symbolic Devices***
  - Pros: No need for physical/virtual device
  - Cons: Inherits cons of symbolic execution

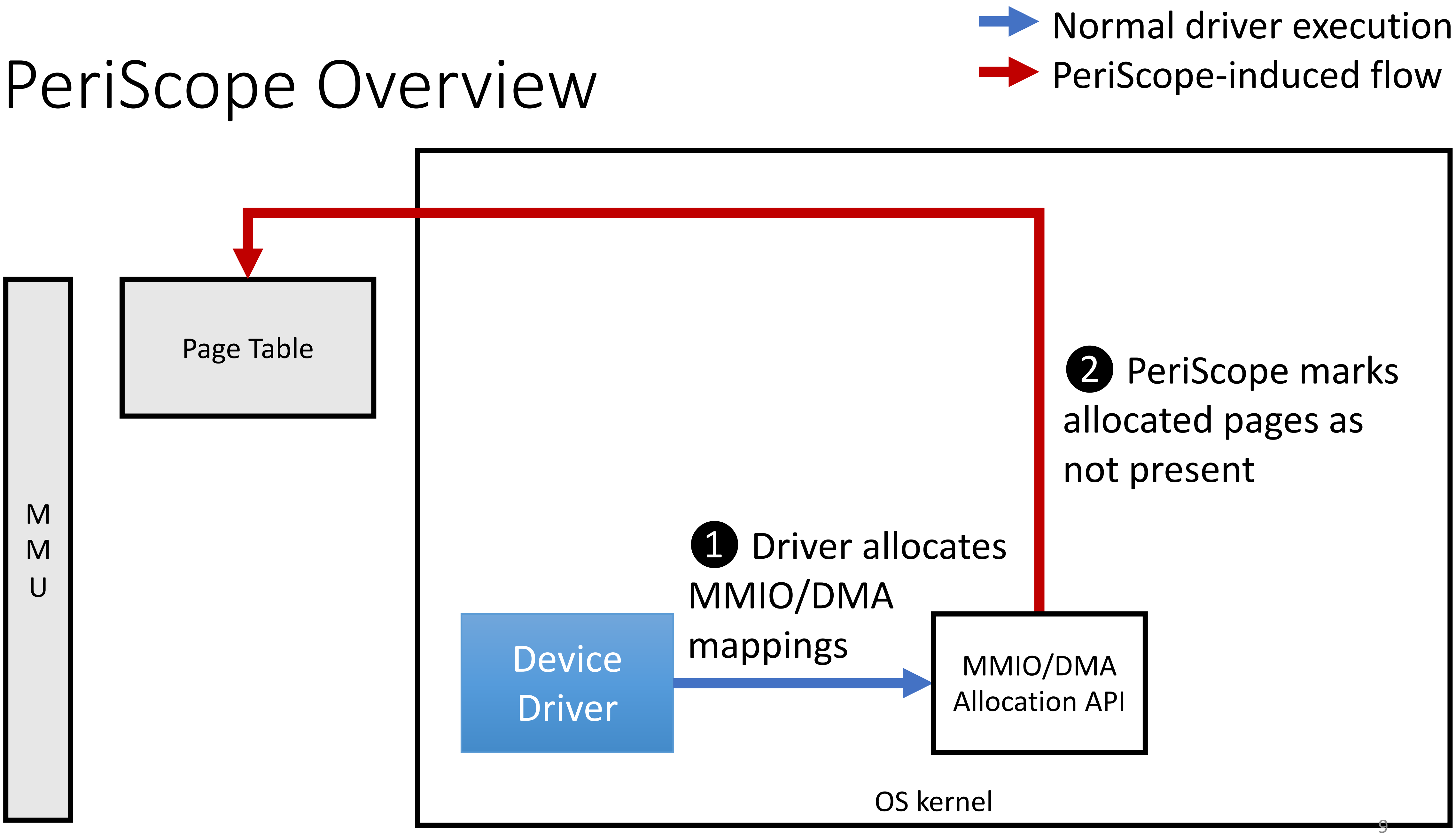


PeriScope – Our Approach

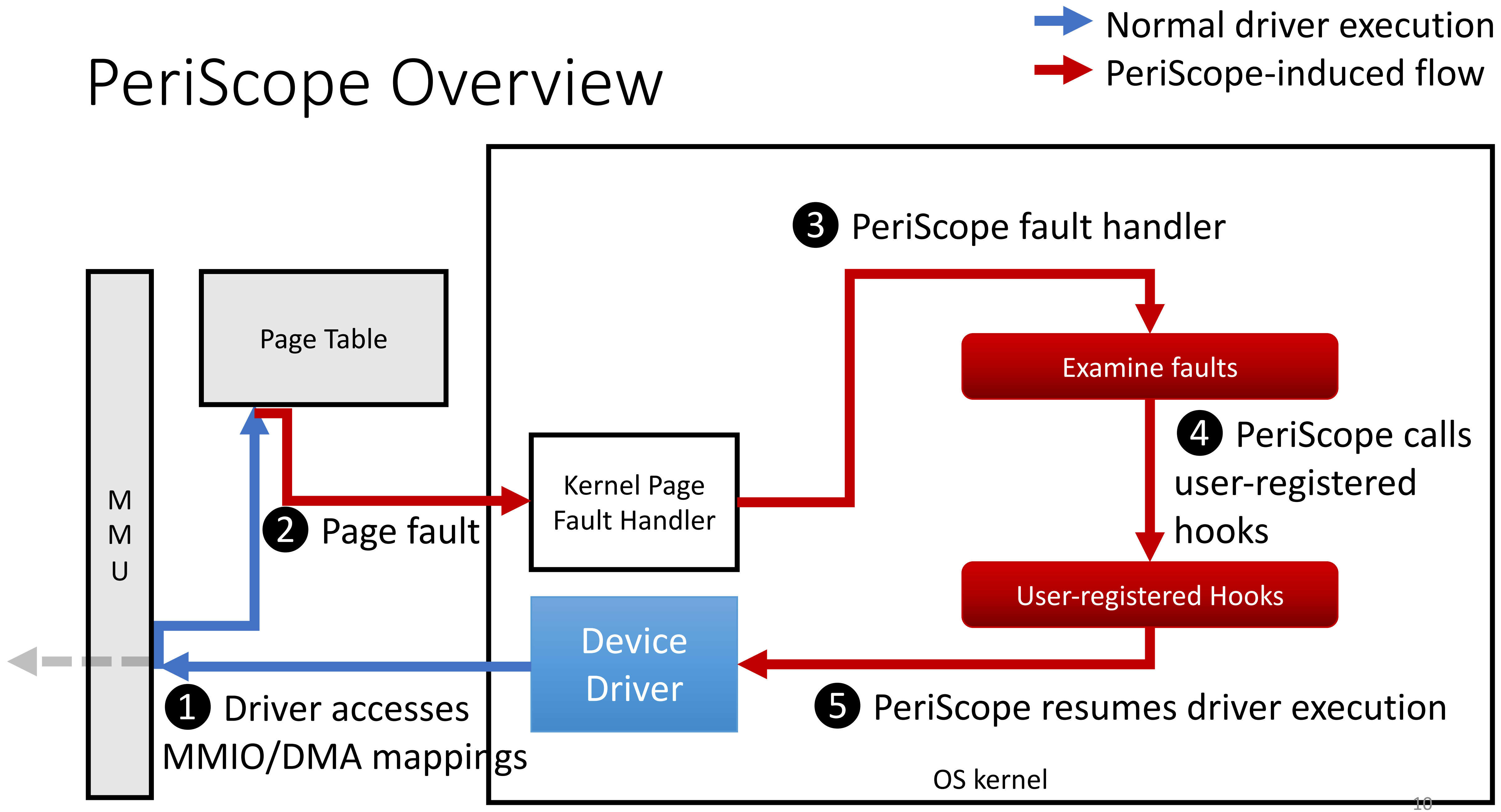
- ***In-kernel, page-fault-based monitoring***
  - Pros: No device-specific/virtualization requirement, Fine-grained monitoring
  - Cons: OS-dependent



PeriScope Overview



PeriScope Overview

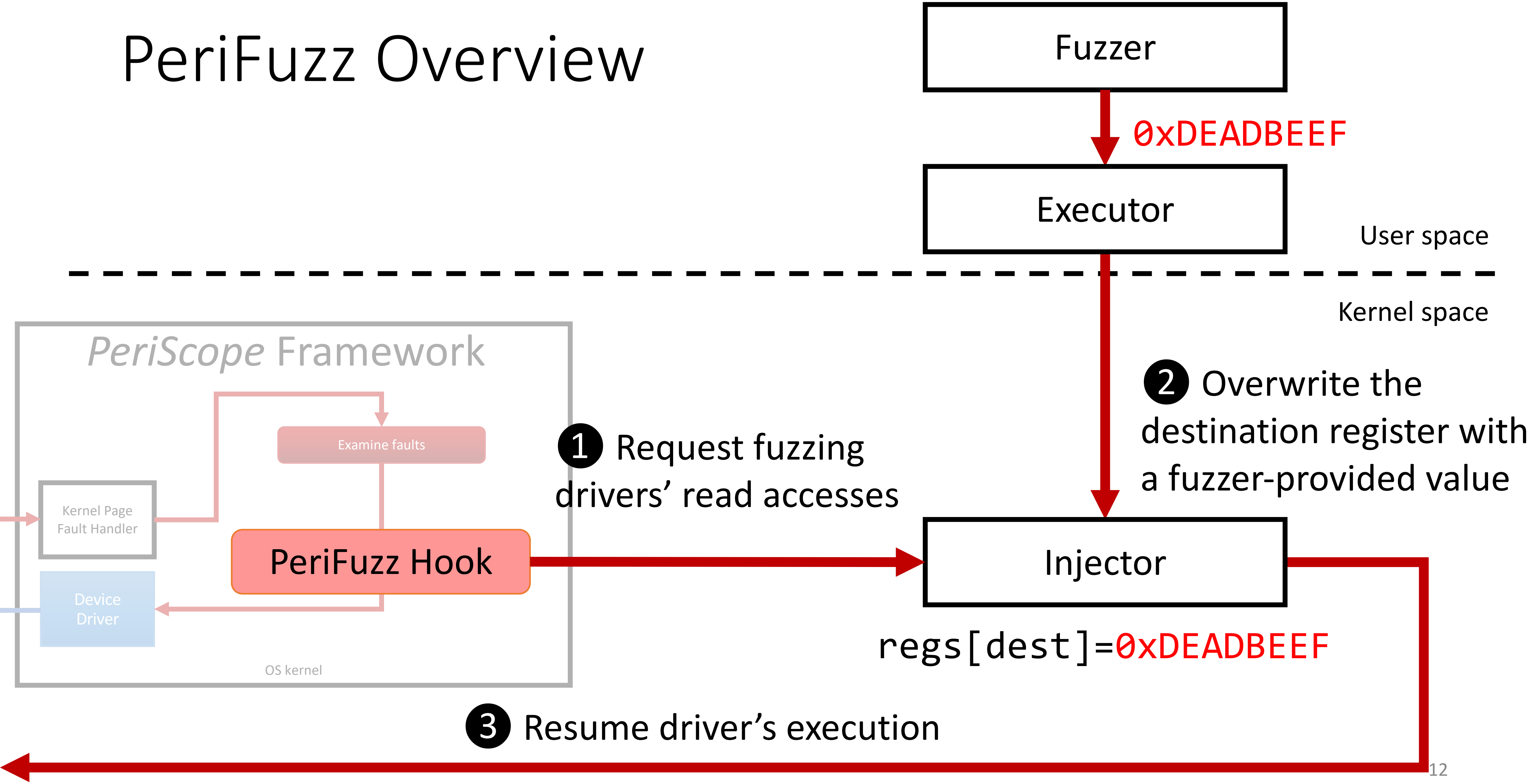


PeriFuzz – Fuzzer for the HW-OS boundary

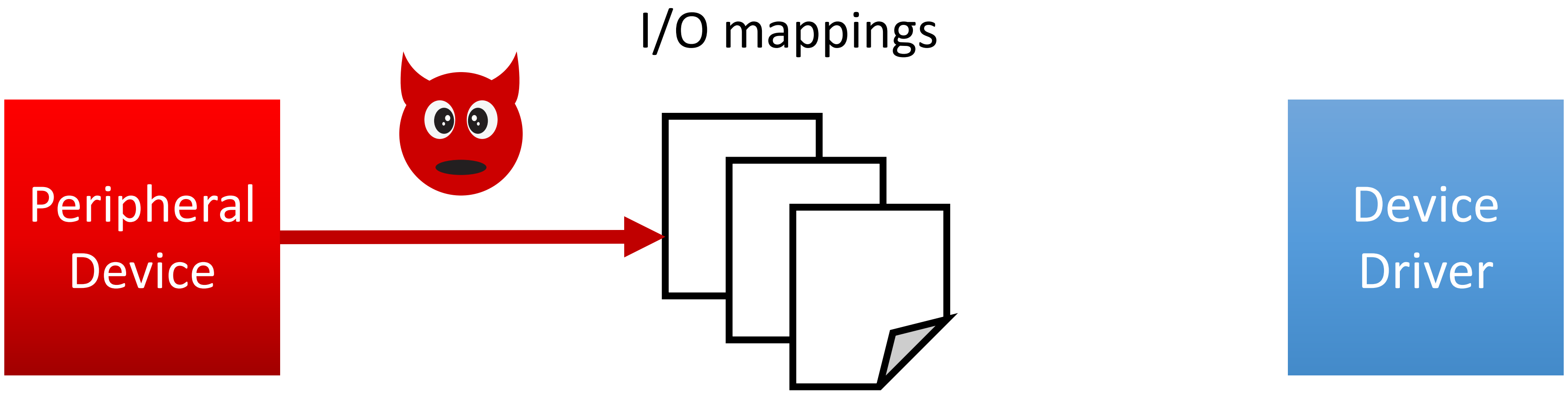
- **Goal**: To find vulnerabilities in drivers reachable from a compromised device

- Therefore, *PeriFuzz* fuzzes **Driver's Read Accesses** to MMIO and DMA mappings

PeriFuzz Overview

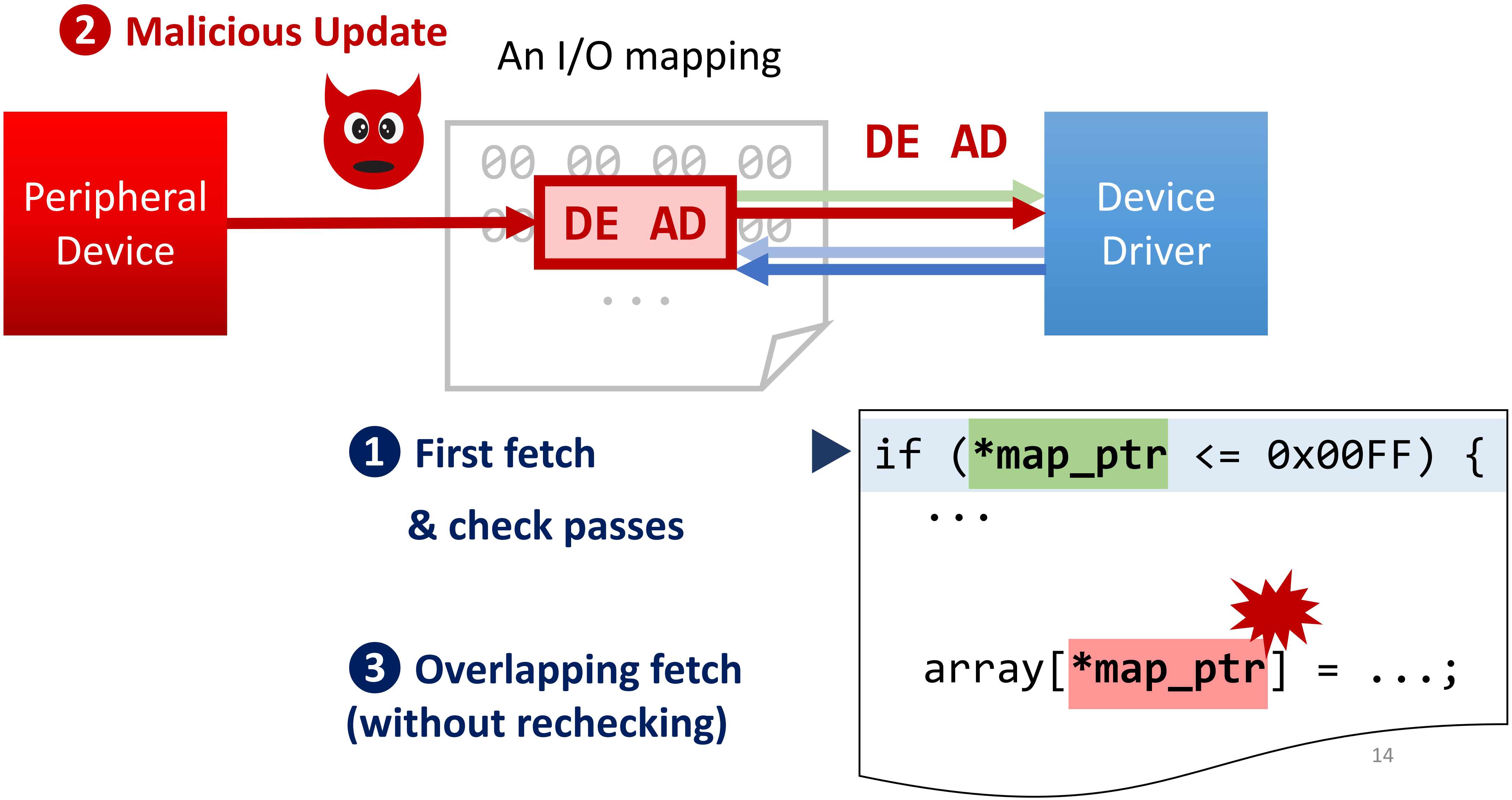


Threat Model Review

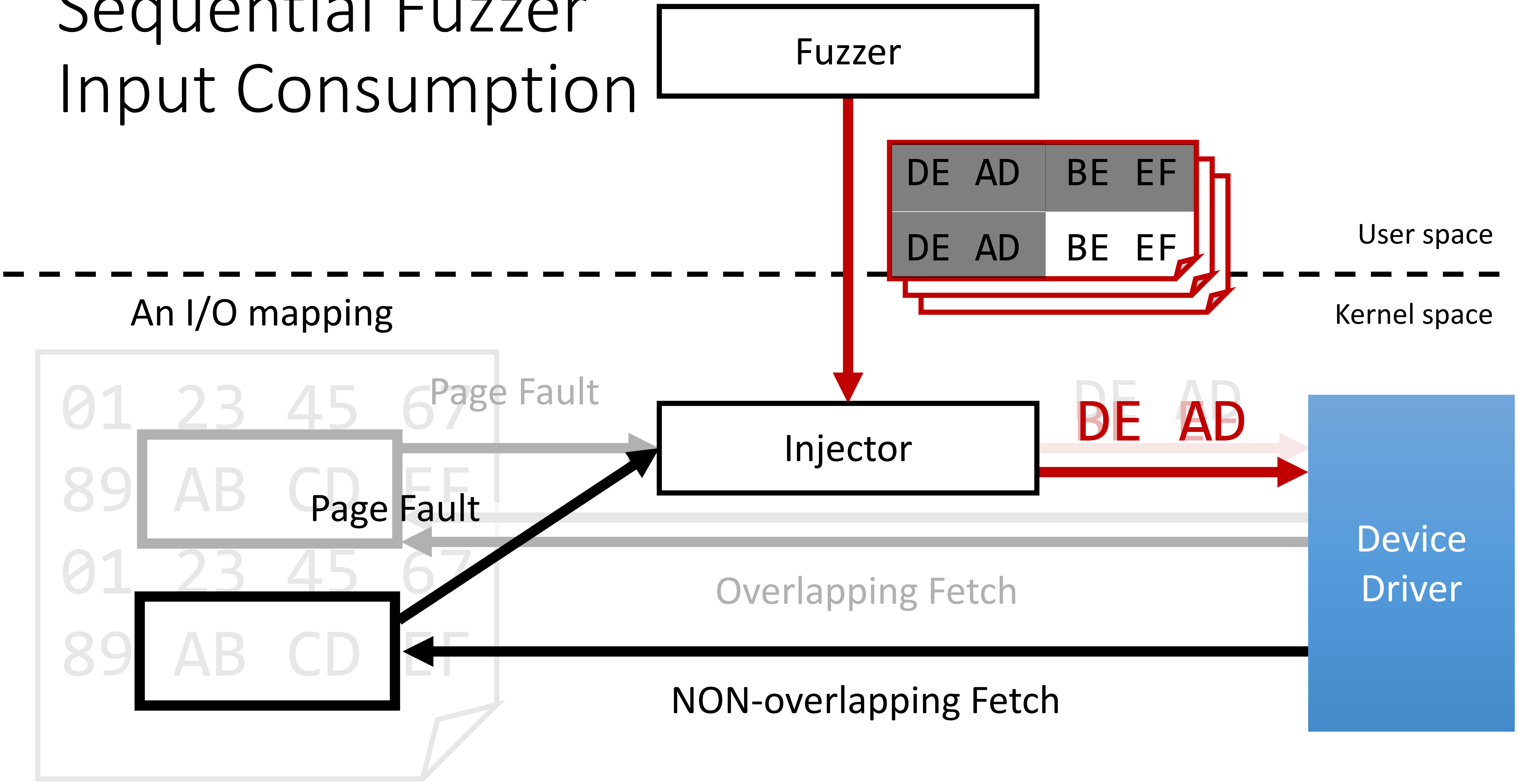


Attacker can write any value to the I/O mappings even multiple times at any time

Potential Double-fetch Bugs in I/O Mappings

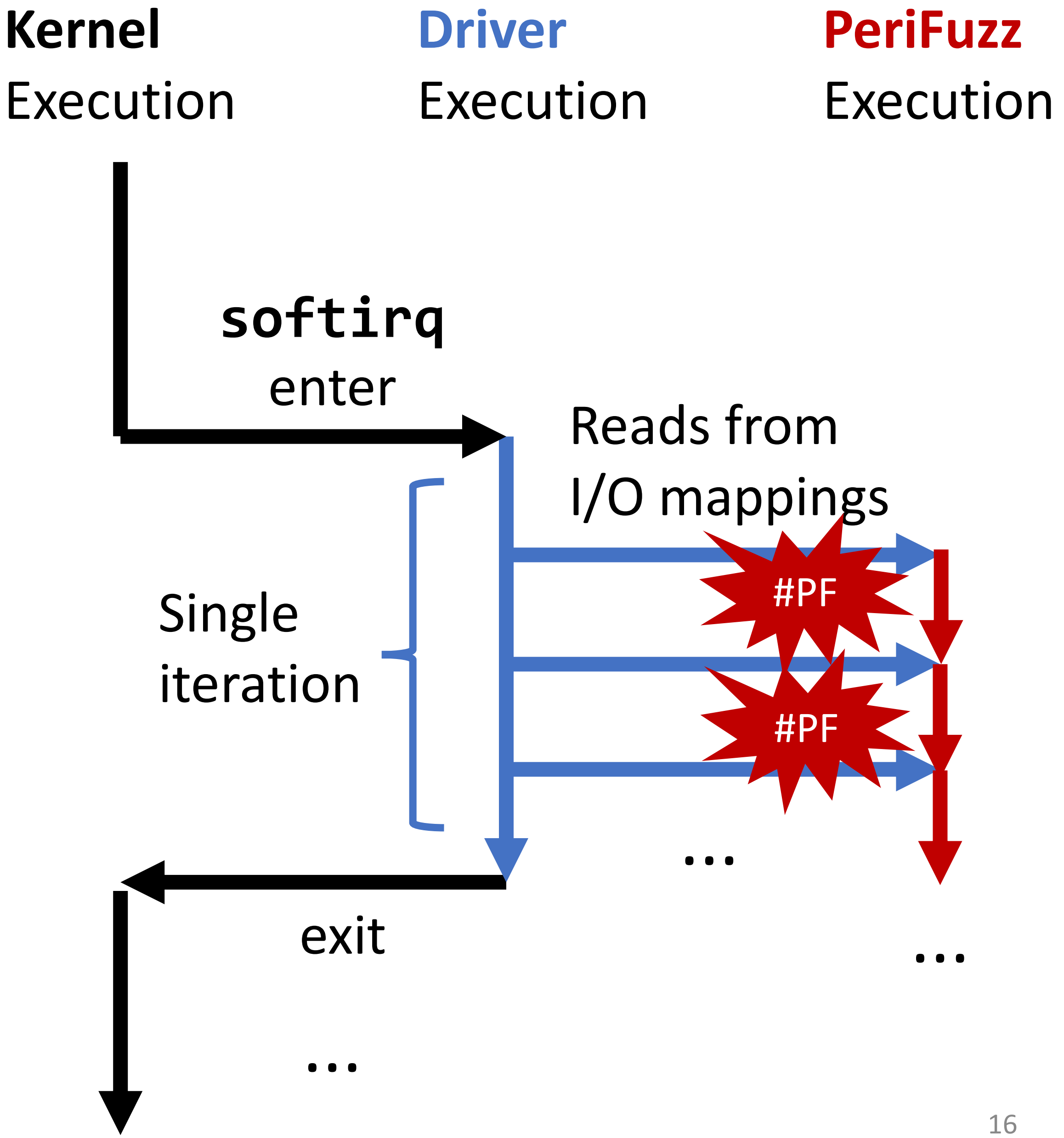


Sequential Fuzzer Input Consumption



Fuzzing Loop

- Each iteration of the fuzzing loop consumes a single fuzzer-generated input
- aligned to the execution of software interrupt (`softirq`) handler's enter & exit
- can have **one or more reads** from I/O mappings.



Prototype Implementation

- Based on Linux kernel 4.4 for AArch64 (Google Pixel 2)
- Ported to 3.10 (Samsung Galaxy S6)
- AFL 2.42b as *PeriFuzz* front-end

Fuzzing Target: Wi-Fi Drivers

1. Large codebase (Qualcomm's: 443,222 SLOC and Broadcom's: 122,194 SLOC)
2. Highly concurrent (heavy use of bottom-half handlers, kernel threads, etc.)
3. Lots of code runs in interrupt & kernel thread contexts (rather than system call contexts)
4. No virtual device implementation available
5. No hypervisor support (EL2 not available in production smartphones)

Bugs Found

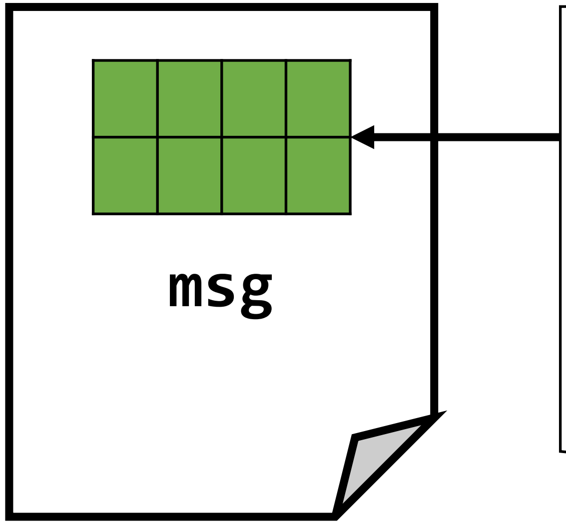
- Different classes of bugs
  - 9 buffer overreads or overwrites
  - 4 double-fetch issues
  - 1 kernel address leak
  - 3 reachable assertions
  - 2 null pointer dereferences

- In total, 15 vulnerabilities discovered
  - 9 previously unknown
  - 8 new CVEs assigned

Double-fetch Bug – Initial Fetch & Check

❶ The driver computes and verifies the checksum of a message

DMA I/O mapping



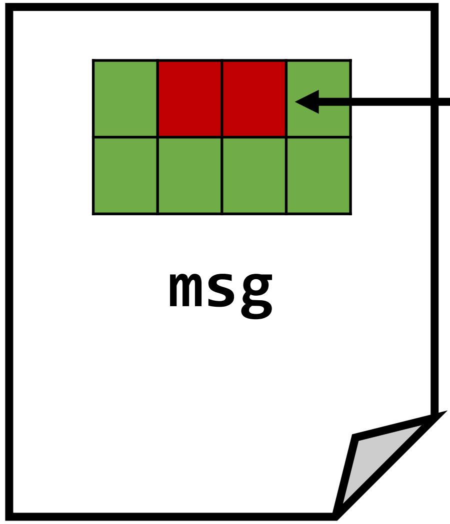
Driver Source Code

```
static uint8 dhd_prot_d2h_sync_xorcsum(...)
  ...
  prot_checksum = bcm_compute_xor32((volatile uint32 *)msg, num_words);
  if (prot_checksum == 0U) { /* checksum is OK */
    if (msg->epoch == ring_seqnum) {
      ring->seqnum++; /* next expected sequence number */
      goto dma_completed;
  ...
```

Double-fetch Bug – Overlapping Fetch & OOB

❷ The driver fetches the same bytes again from msg

DMA I/O mapping



Driver Source Code

```
ifidx = msg->cmn_hdr.if_id;
...
ifp = dhd->iflist[ifidx];
```

Overlapping fetch (fuzzed)

Out-of-bounds access

```
Unable to handle kernel paging request at virtual
address 2f6d657473797337

Kernel panic - not syncing: Fatal exception in
interrupt
```

Kernel Address Leak (CVE-2018-11947)

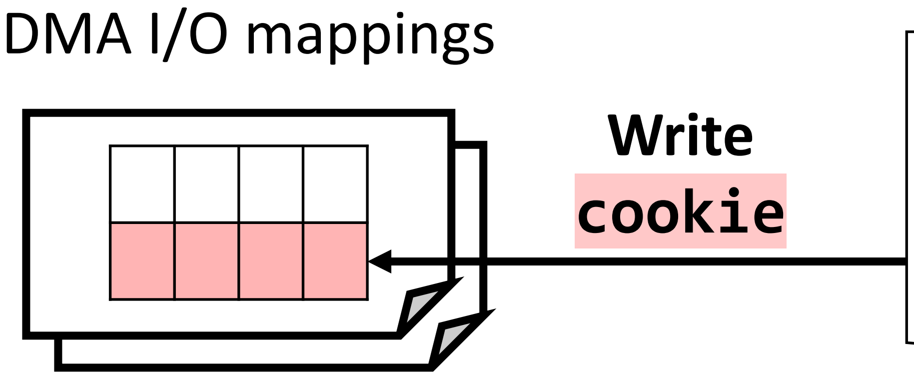
```
Unable to handle kernel paging request at virtual address
17000000d7ff0008

Kernel panic - not syncing: Fatal exception in interrupt
```

Symptom: A fuzzed value provided by *PeriFuzz* was *directly* being dereferenced.

Kernel Address Leak (CVE-2018-11947)

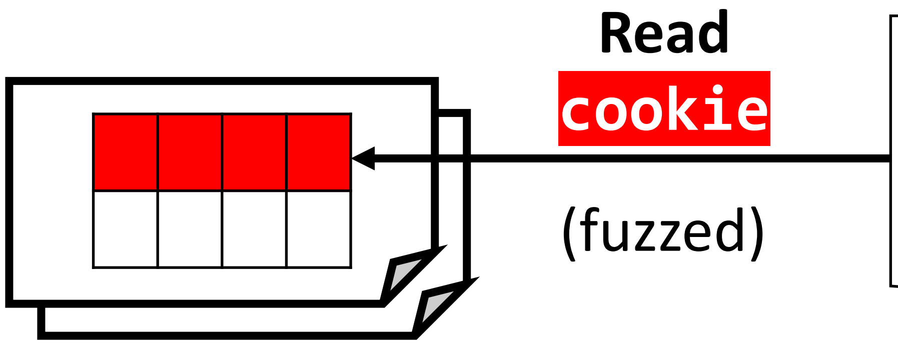
❶ Driver sends a kernel pointer to the device



Driver Source Code

```
non_volatile_req = qdf_mem_malloc(sizeof(*non_volatile_req));
...
// use pointer as cookie (which is later sent to the device)
cookie = ol_txrx_stats_ptr_to_u64(non_volatile_req);
...
```

❷ Device sends the cookie back, which is then dereferenced by the driver




```
req = ol_txrx_u64_to_stats_ptr(cookie);
...
req->... // A value read from I/O mapping is dereferenced
```

Fuzzing Throughput

- Fuzzing throughput is about 7~24 inputs/sec depending on the nature of the I/O mapping being fuzzed.
- The number of page faults is the main contributor.
- We expect an improvement of at least 2x-3x with further optimization. (Details in the paper)

Phone/Driver	I/O Mapping	Peak Throughput (# of test inputs/sec)
Pixel 2 - QCACLD-3.0	QC1	23.67
	QC2	15.64
	QC3	18.77
	QC4	7.63
Galaxy S6 - BCMDHD4358	BC1	9.90
	BC2	14.28
	BC3	10.49
	BC4	15.92

cf) On Pixel 2, Syzkaller achieves on average 24 program executions per second (max: ~60).
(1 proc ADB-based configuration measured for a 15-min period)

Future Work

- Minimizing the impact of shallow bugs
  - All bugs found in less than 10000 inputs
  - Shallow bugs frequently hit, which causes system restarts (reboot takes 1 min)
  - We had to manually disable subpaths rooted at bugs already found
- Improving throughput
  - Slower than, for example, typical user-space fuzzing
  - Possible optimizations and trade-offs outlined in the paper

Conclusion

- Remote peripheral compromise poses a serious threat to OS kernel security.
- PeriScope and PeriFuzz are practical dynamic analysis tools that can analyze large, complex drivers along the hardware-OS boundary.
- PeriScope and PeriFuzz are effective at finding vulnerabilities along the HW-OS boundary.
  - Memory overreads/overwrites, address leak, null pointer dereferences, reachable assertions, and double-fetch bugs

Q & A

Thank you!

Contact

Dokyung Song

Ph.D. Student at UC Irvine

dokyungs@uci.edu