




IMPULP: A Hardware Approach for In-process Memory Protection via User-Level Partitioning

Yangyang Zhao, Mingyu Chen, Yuhang Liu, Zonghao Yang,
Xiaojing Zhu, Zonghui Hong, Yunge Guo



Zhao YY, Chen MY, Liu YH et al. IMPULP: A hardware approach for in-process memory protection via user-level partitioning. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 35(2): 418–432 Mar. 2020. DOI 10.1007/s11390-020-9703-2



CENTER FOR ACS,
INSTITUTE OF COMPUTING TECHNOLOGY,
CHINESE ACADEMY OF SCIENCES.

Our proposal: IMPULP

2

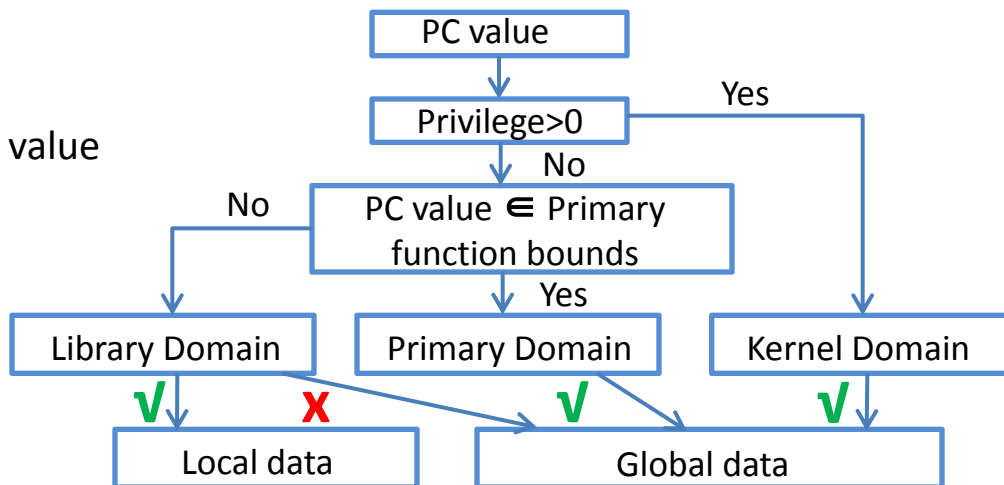
- We propose IMPULP, a hardware approach for in-process memory protection
 - divides user-level security domains by the range of corresponding instruction addresses(equals to the value of program counter);
 - User code with different instruction addresses(PC values) can access different ranges of memory address space.
- Efficiency advantages
 - The switch of different security domain is executed with slightly modification of a program using APIs;
 - When each instruction is executed, the modified CPU pipeline in hardware automatically configures and checks the boundaries for each instruction;
 - The process with illegal accesses will be halted timely.



Key Point : User-Level Memory Protection

3

- Divide Security Domain
 - User-Level Partition
 - with **PC (program counter)** value



- For Primary Function

- No Restriction

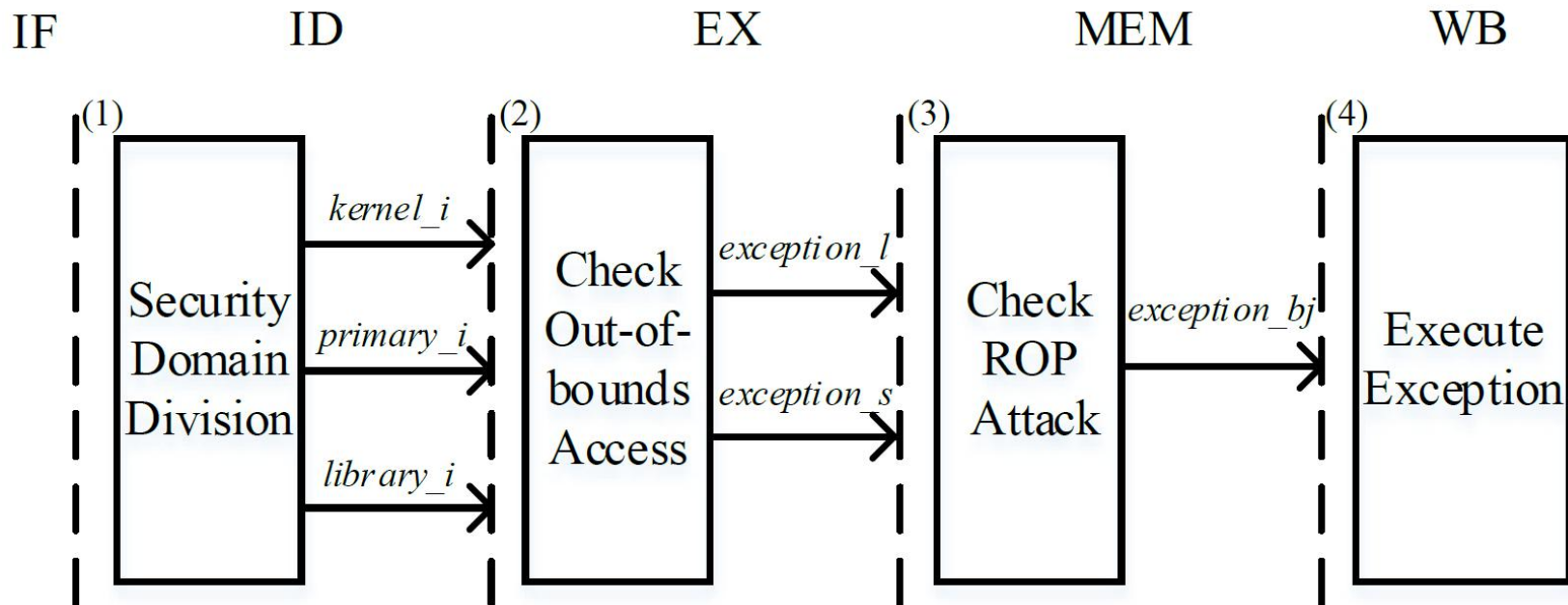
- For Library Function

- 1. Configure the range of accessible memory addresses **before function call**.
- 2. Check out-of-bounds memory accesses with hardware support **at runtime**.



Hardware-assisted: Pipeline modification

4



Conclusion

5

- We propose the **IMPULP** approach to **realize in-process data isolation**. IMPULP classifies user process into reliable primary functions and untrusted library functions, endowing different privileges to the two parts.
- Experimental results show that IMPULP could prevent in-process abuse attacks such as buffer overflow and memory leakage. The **runtime overhead** of IMPULP is less than 0.2%, which is **negligible**. The resource overhead is less than 5.5% for hardware modification.
- Future Work
 - Extension to other platforms(such as Intel and ARM) with more memory access related instructions
 - Support nested function calls and conventional CFI methods
 - Extension to other security levels

