CS 578: Cyber-security
Part III: Isolation

Sanghyun Hong
sanghyun.hong@oregonstate.edu

Oregon State University

SAIL
Secure AI Systems Lab

ANNOUNCEMENT

- HW3 will be out by next Monday
- 5/12 and 14 lectures will be online (SH’s business travel)
  - On those dates, in-class presentations will also be online

PROBLEM: VULNERABLE CODE IN C

- Many security vulnerabilities
  - Buffer overrun, use-after-free
  - Return to LibC
  - Malicious code injection
  - …
- Unsafe memory operations
  - One can overwrite function pointers
  - One can overwrite a return address
  - …

PROBLEM: VULNERABLE CODE IN C

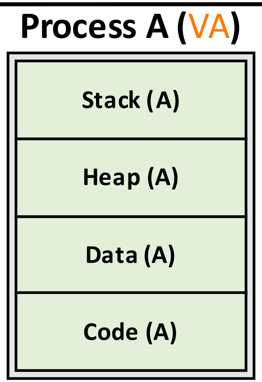
- Untrusted software modules
  - Modern OSes have components and modules developed by 3rd parties
  - Applications include modules or libraries, untrusted
  - Or Internet browsers, running 3rd-party extensions
  - … (more)

- They can do unsafe memory operations
  - Modules, components, or libraries will run in an application's address space
  - Those components can
    - Overwrite the data
    - Steal confidential data
    - Call malicious functions or call functions with malicious arguments
    - … All efforts in subverting a target system

ISOLATION IS THE KEY IN COMPUTER SYSTEMS SECURITY

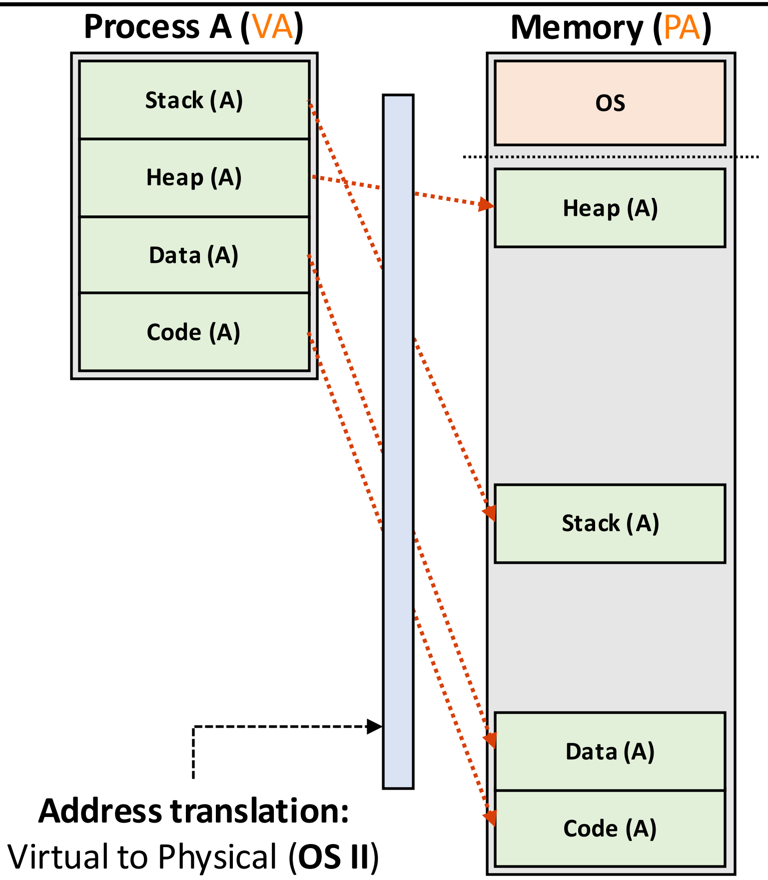
EXAMPLE: PROCESS ISOLATION

- Process segments
  - Code segment
  - Data segment
  - Heap segment
  - Stack segment



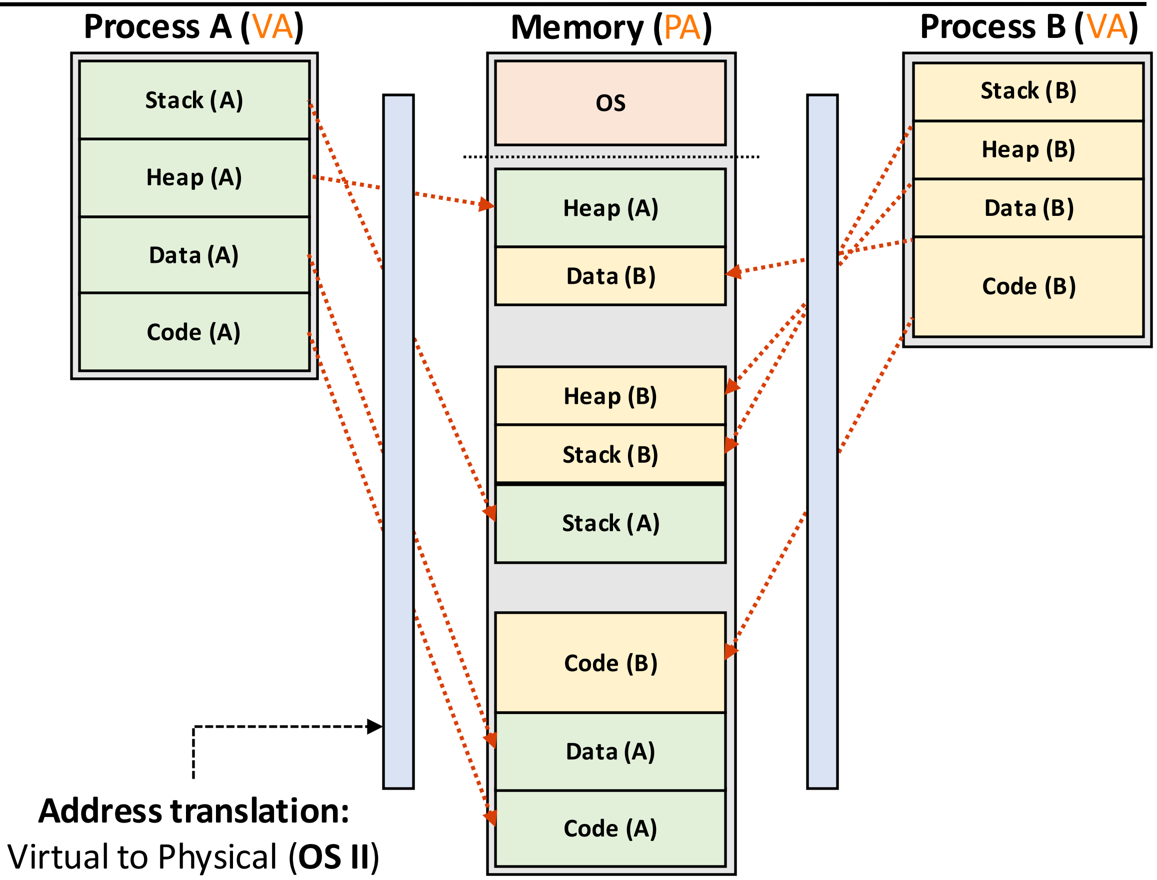
EXAMPLE: PROCESS ISOLATION

- Process segments
  - Code segment
  - Data segment
  - Heap segment
  - Stack segment



EXAMPLE: PROCESS ISOLATION

- Process segments
  - Code segment
  - Data segment
  - Heap segment
  - Stack segment



EXAMPLE: PROCESS ISOLATION

- Process **isolation**
  - **Definition:** Prevent Process A from reading/writing to Process B
  - Why?
    - Security reasons (e.g., data breach, system crash, …)
    - Management reasons (e.g., easy to control, …)
  - What happens if we access the other process' memory
    - **Segmentation fault**

EXAMPLE: PROCESS ISOLATION

- Does it solve the problem?
  - Well… probably no
    - What if the untrusted modules, components, are libraries closely coupled in an app?
    - What if those 3rd-party components are running within a process' memory space

STRAWMAN SOLUTION

- **Two separate processes!**
  - **Method:**
    - A process only runs trusted components
    - The other process only runs un-trusted components
  - **Downside:**
    - Implementation overhead to programmers
    - Performance overhead due to many IPC calls (CTX switch)

- **Hole punching (Link)!**
  - **Definition** (from computer networking)**:**
    - A technique that allows two or more parties to communicate directly each other
  - **Downside:**
    - Potentially ignore the security mechanisms (e.g., firewalls)
    - Potentially increase overheads to manage such connections separately

ISOLATION IS THE KEY IN COMPUTER SYSTEMS SECURITY - SANDBOXING AND TRUSTED ENCLAVE

Software-based fault isolation (SFI)

- SFI Goals
  - To make the isolation cheap
  - To use a single address space:

- Technical approaches
  - Run untrusted code, modules, or libraries in the same address space as trusted code
  - Run untrusted code in sandbox

- Key idea
  - One can add instructions before memory writes and jumps
  - Those instructions inspect the target addresses to constrain their behaviors

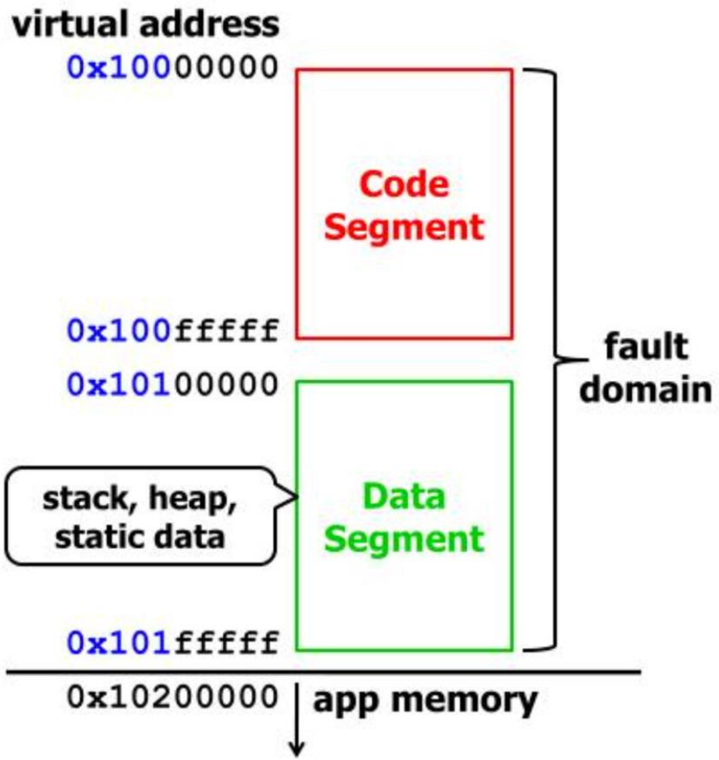
SOFTWARE-BASED FAULT ISOLATION

- Unit of operations: fault domain
  - SFI puts untrusted code within a fault domain
  - The fault domain is in the same address space as trusted code

- The fault domain has
  - Unique ID
  - Code segment
  - Data segment
  - Segment ID: unique high-order bits for a segment

SOFTWARE-BASED FAULT ISOLATION

- Unit of operations: fault domain – an example
  - Segment ID are 12 high-order bits
  - Separate segments for code and data

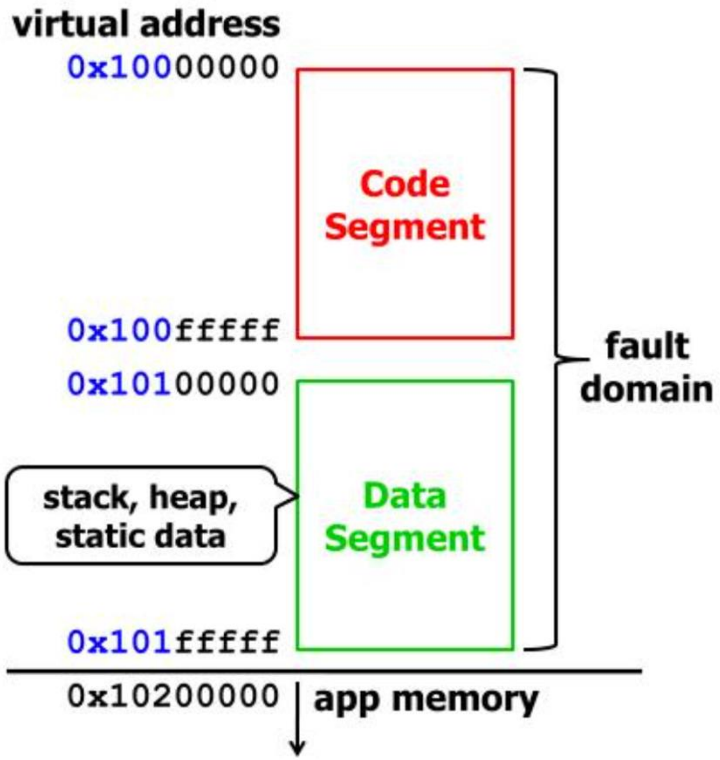


Software-based Fault Isolation

- Sandboxing memory: *segment matching*
  - *Jump* within its fault domain segments
  - *Write* within its fault domain segments

- It supports two memory addresses
  - Direct, e.g., jmp 0xdeadbeef
  - Indirect, e.g., store %ebp %eap

- Protection
  - Direct: check the computed address



SOFTWARE-BASED FAULT ISOLATION

- Sandboxing memory: *segment matching*
  - *Jump* within its fault domain segments
  - *Write* within its fault domain segments

- It supports two memory addresses
  - Direct, e.g., jmp 0xdeadbeef
  - Indirect, e.g., store %ebp %eap

- Protection
  - Direct: check the computed address
  - Indirect: use four dedicated registers
    - The code and data segment addresses
    - The segment shift amount
    - The segment ID

```
STORE R0, R1     ; write R1 to Mem[R0]
```

↓

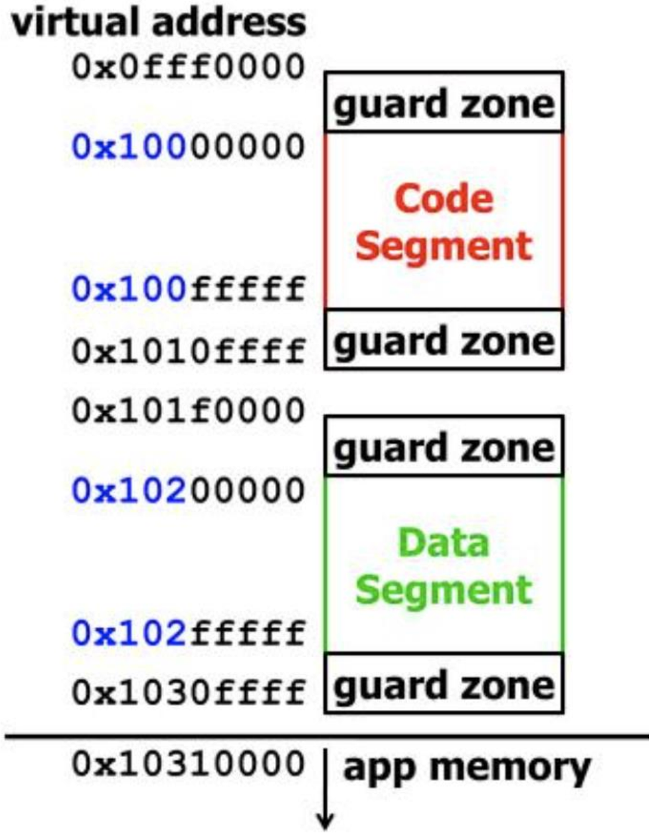
```
MOV Ra, R0       ; copy R0 into Ra
SHR Rb, Ra, Rc   ; Rb = Ra >> Rc, to get segment ID
CMP Rb, Rd       ; Rd holds correct data segment ID
BNE fault        ; wrong data segment ID
STORE Ra, R1     ; Ra in data segment, so do write
```

SOFTWARE-BASED FAULT ISOLATION

- Sandboxing memory: *segment matching*
  - *Jump* within its fault domain segments
  - *Write* within its fault domain segments

- It supports two memory addresses
  - Direct, e.g., jmp 0xdeadbeef
  - Indirect, e.g., store %ebp %eap

- Performance optimization 1: guard-zones
  - Use compiler-base approaches
  - Use instructions of *register+offset*
  - Offsets are +/-64K, e.g., in MIPS



SOFTWARE-BASED FAULT ISOLATION

- Sandboxing memory: *segment matching*
  - *Jump* within its fault domain segments
  - *Write* within its fault domain segments

- It supports two memory addresses
  - Direct, e.g., jmp 0xdeadbeef
  - Indirect, e.g., store %ebp %eap

- Performance optimization 2: stack pointer
  - Avoid sandboxing all the read/write operations to SP
  - Stack pointer is read more often than its written
  - Sandbox the process of writing the stack pointer (it's always safe)
  - Reduces the number of instructions sandboxed

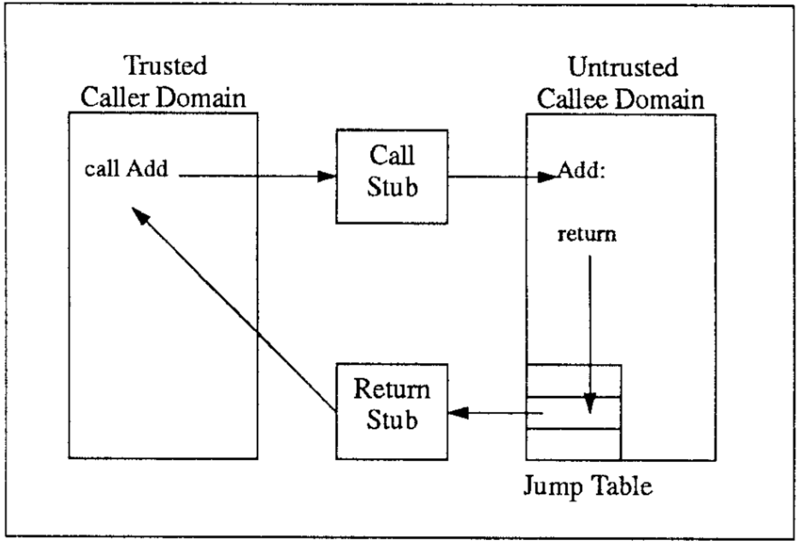
SOFTWARE-BASED FAULT ISOLATION

- Sandboxing memory: *segment matching*
  - *Jump* within its fault domain segments
  - *Write* within its fault domain segments

- Data sharing
  - Do it on the virtual address spaces
  - Read-only sharing
  - Virtual address *aliasing*
    - The lower bits are the same in the virtual addresses of different segments
    - Once the untrusted code accesses a shared object,
      it first translates the shared addresses into the corresponding addresses
      within the fault domain

SOFTWARE-BASED FAULT ISOLATION

- Sandboxing memory: *segment matching*
  - *Jump* within its fault domain segments
  - *Write* within its fault domain segments



- Data sharing
  - Do it on the virtual address spaces
  - Read-only sharing
  - Virtual address *aliasing*

- RPC for cross-fault domain communication: *jump table*
  - In the read-only region
  - A collection of code addresses written by trusted parties
  - Only called via trusted call and return stubs

SANDBOXING EVALUATION

- Encapsulation overhead
  - 4.3% execution time overhead across different benchmarks

ISOLATION IS THE KEY IN COMPUTER SYSTEMS SECURITY - SANDBOXING AND TRUSTED ENCLAVE