

Terms:

SFI - Software Fault Isolation

CFI – Control Flow Isolation

Basic block property – Execution of a basic block always starts from its beginning

Summary of last class:

Context switching between 2 processes which communicate heavily with each other is very costly. Therefore it is advisable that the 2 processes run in the same address space. But since these 2 processes may not trust each other or trust each other partially, it is necessary that these processes are still isolated from each another.

SFI says that every memory access should be done after a bounds check.

Basic block property is very helpful to assure Fault isolation.

A system in which every basic block has a first instruction which specifies the 'tag' for the basic block and ensures that any jump instruction is within the memory boundaries set by 'tag' can allow isolation of security domains which lie in the same address space.

Class – 6th Apr 2006

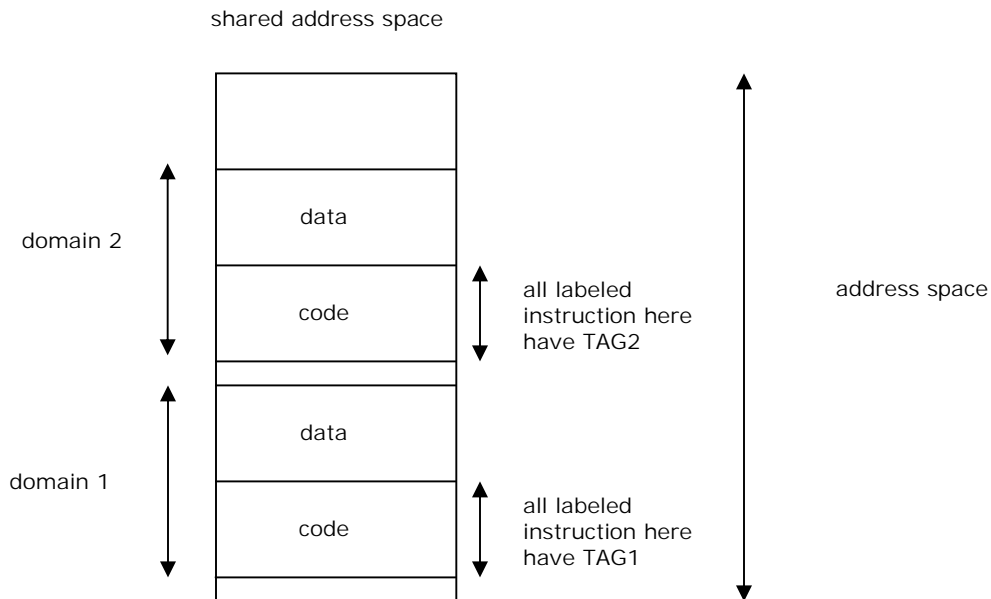
CFI

The CFI assures the basic block property.

- Every basic block executes from its beginning.

SFI

Restricts one module to the subset of the address space.



In last class we have seen that by adding check instructions before memory access, we can build a system that will isolate one security domain from another.

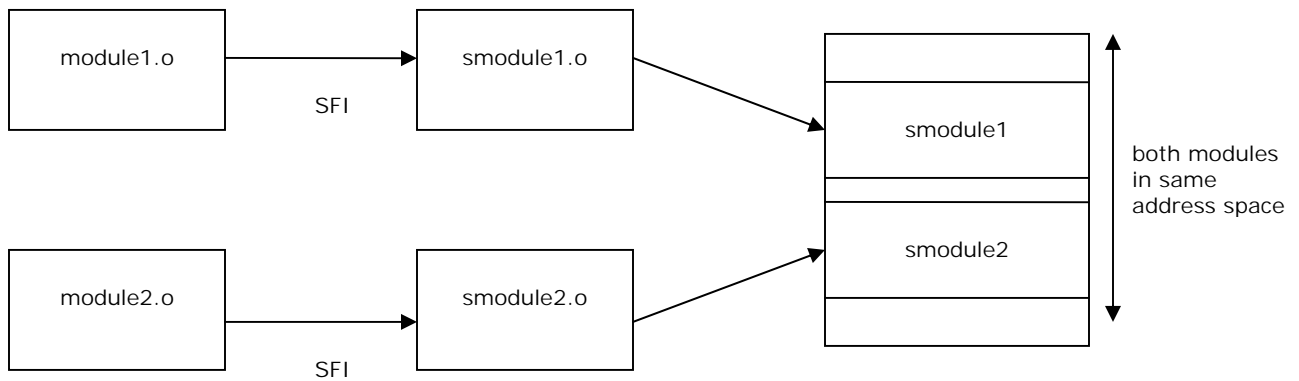
What if attacker in domain 1 sends malicious data to domain 2 in order to do something malicious at domain 2?

Extending the concept of isolation of various domains, we can isolate the code segment from data segment so that no code the data segment is executed.

What if a domain rewrites its own code?

- The code can be made read only
 - limit the code to write only new data and not the code
- The code and data can be further classified as 'domain1a' and 'domain1b'

Actual working of the SFI model



SFI is a decompiler which inserts code like
setlib 0xc1, eax
cmp [eax], TAG
jne ABORT
before any 'jmp eax' instruction

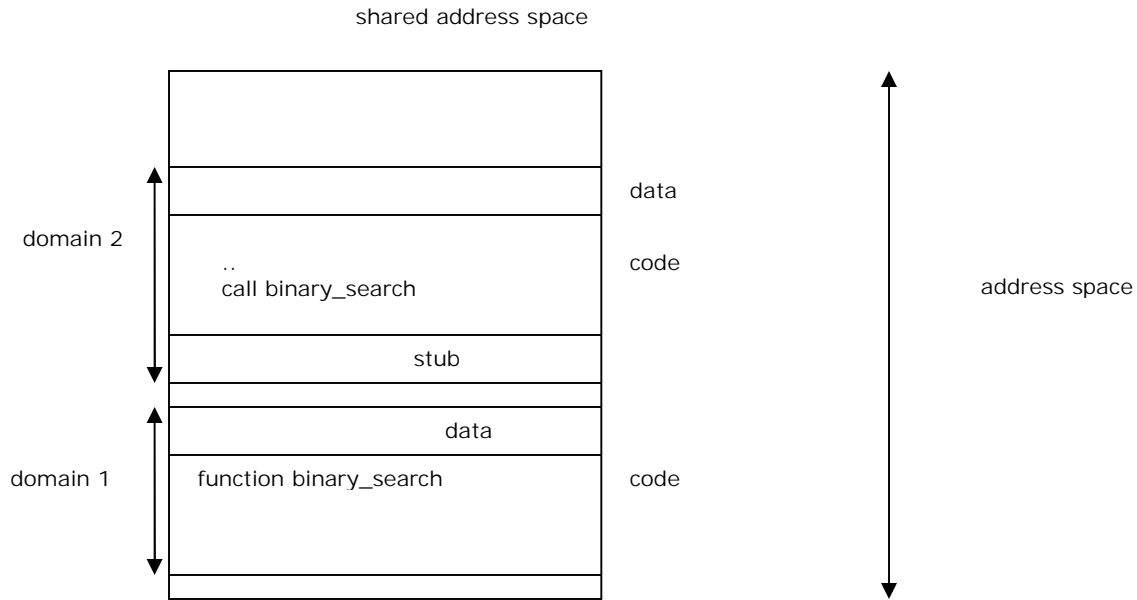
Proof by induction:

- All code in basic block always executes from 1st instruction of the basic block.
- A module can exec only the code in its own security domain.

So using SFI, we have achieved following:

- basic block property
- no cross domain read / writes / jumps

But how to achieve inter-domain communication?



Cross domain calls are made via stub

- stub is called from domain2
- stub lies in code segment of domain2
- stub can read / write data in domain 2
- stub can read / write data in domain 1
- stub can jump to domain 1 (and no other code in domain 2 can jump to other domain)

Performance :

- Very low overhead for frequent domain crossing
- Lot better than function call

	PIPE	SFI	Function call
	200 micro sec	1 micro sec	0.1 micro sec

-flexible security policy (does not depend on H/W or OS)

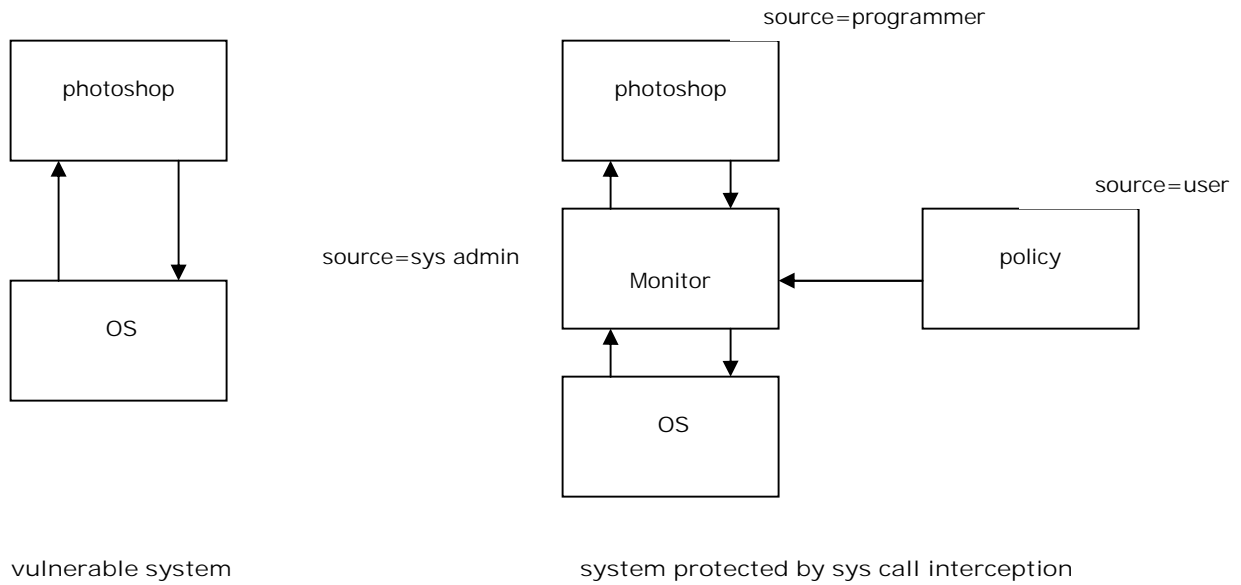
System call interposition

Motivation:

Consider the following problem:

I have downloaded a pirated copy of photoshop and suspect that somebody may have inserted a trojan horse inside it.

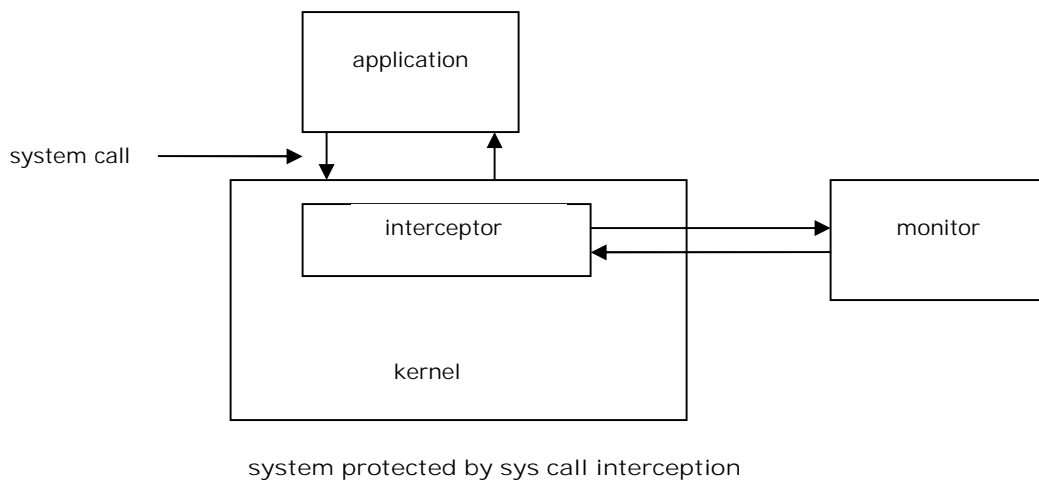
As a security step, one must ensure that photoshop does not write to any file under c:\Windows



The monitor keeps track of what system calls are made and is that permissible or possible for the application making the system calls.

The monitor can be a part of the kernel so that there are no context switches between the kernel space and the user space for system call interposition. But a monitor with code of about 3 – 4k lines of code would make the kernel bulky.

Therefore we can have the monitor consist of 2 parts – the interceptor and the monitor.



Race condition:

