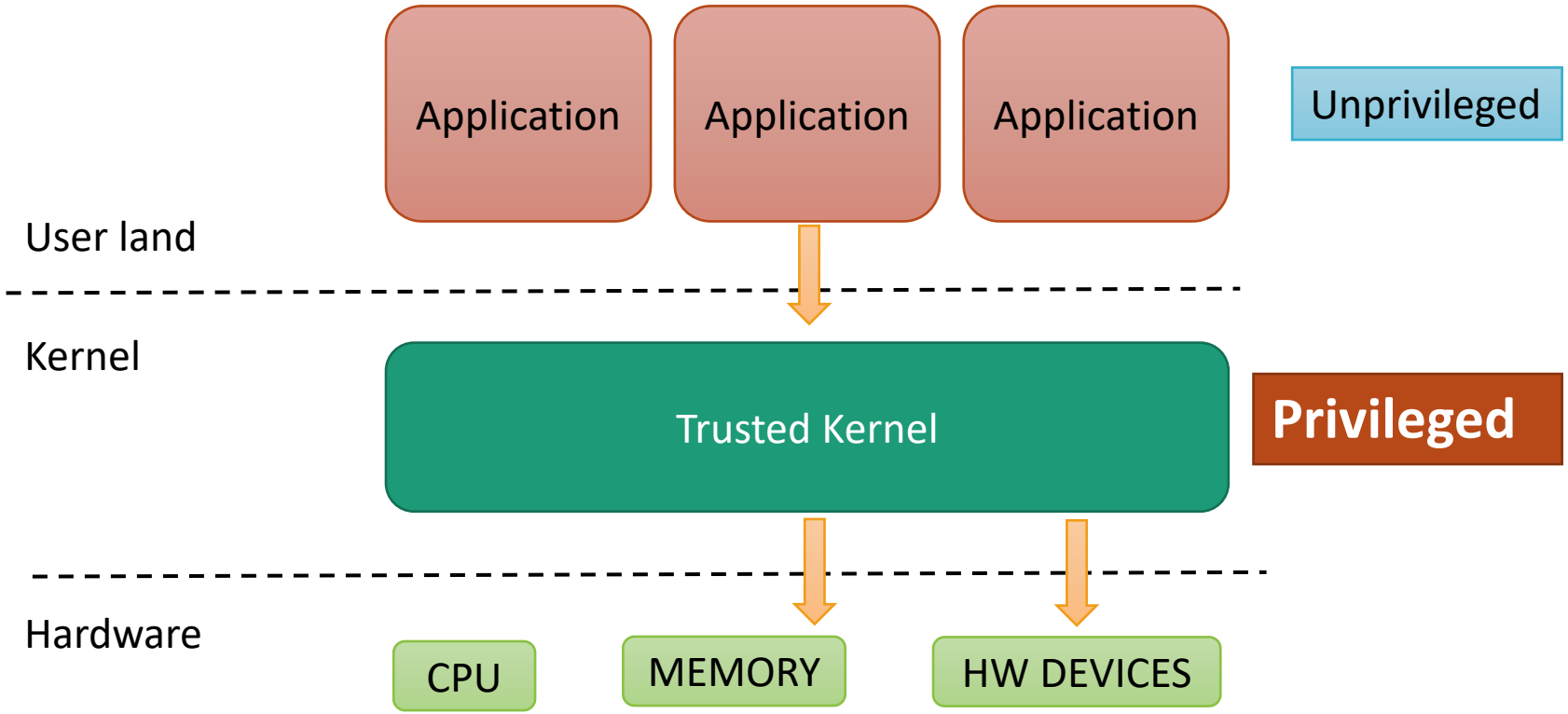
Securing the OS

CS-576 Systems Security

Instructor: Georgios Portokalidis

Fall 2018

Why Attack the OS/Kernel



Kernel Code Has Bugs Too

Overflows: Writing beyond the end of a buffer

Underflows: Writing beyond the beginning of a buffer

Use-after-free: Using memory after it has been freed

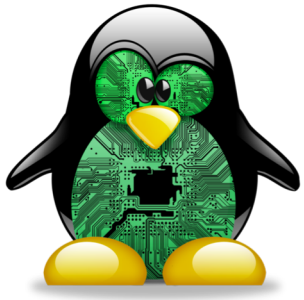
Integer overflows

Null pointer dereferences: Using NULL pointers

…

Kernel vs Applications

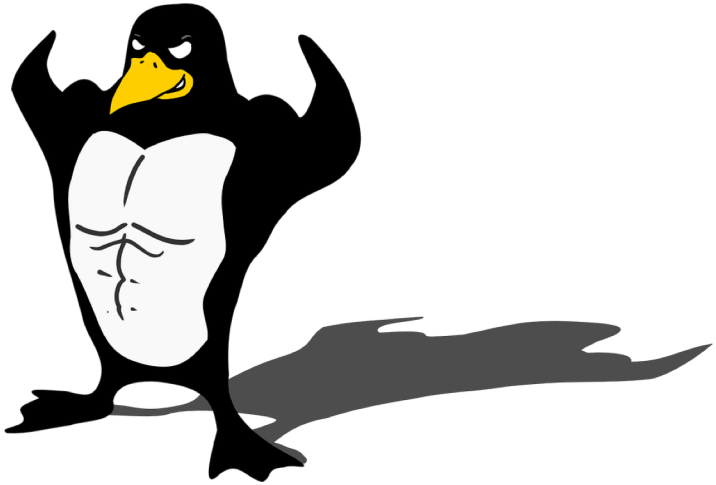
Applications



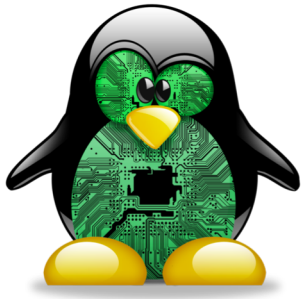
Kernel

Kernel vs Applications

ASLR, W^X, heap & stack protection, CFI

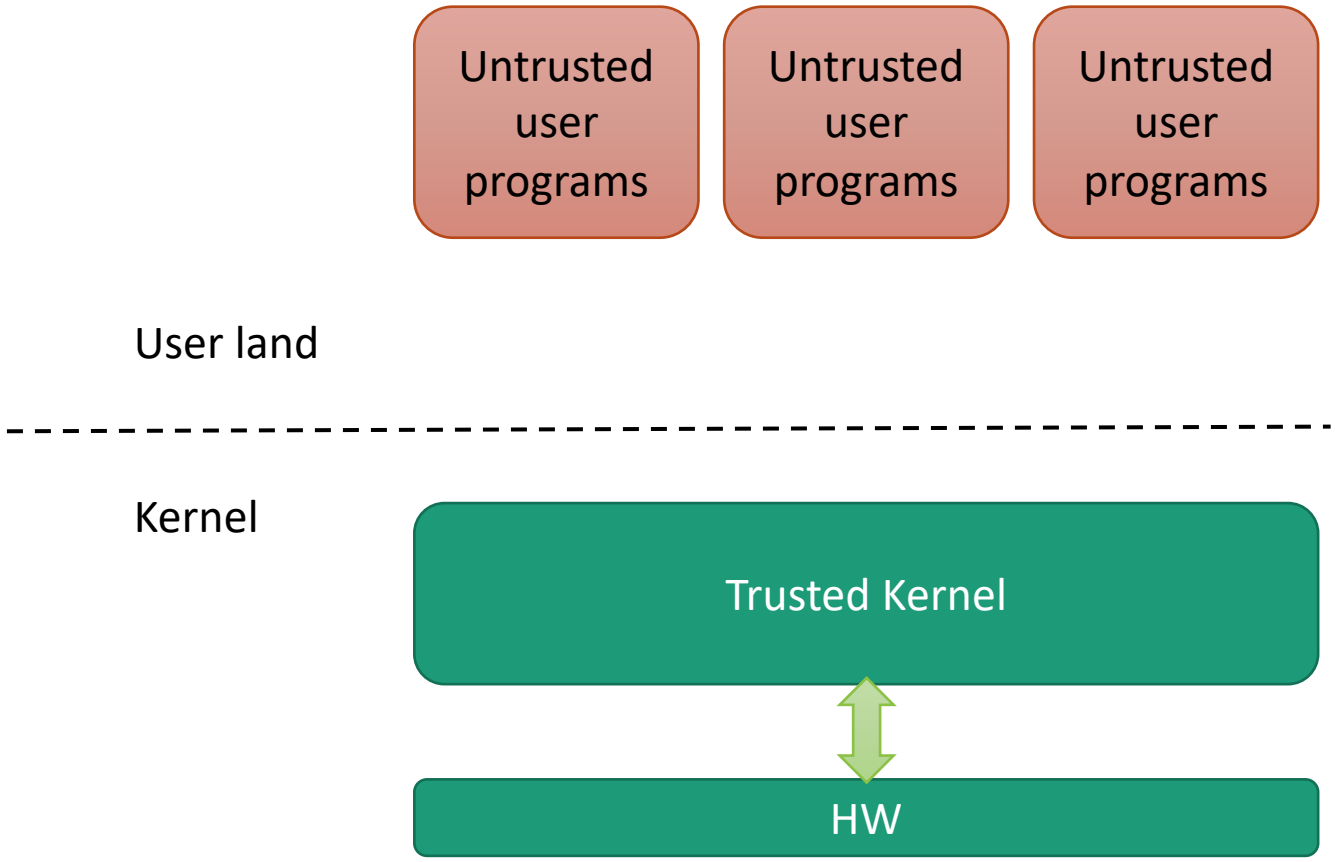


Applications

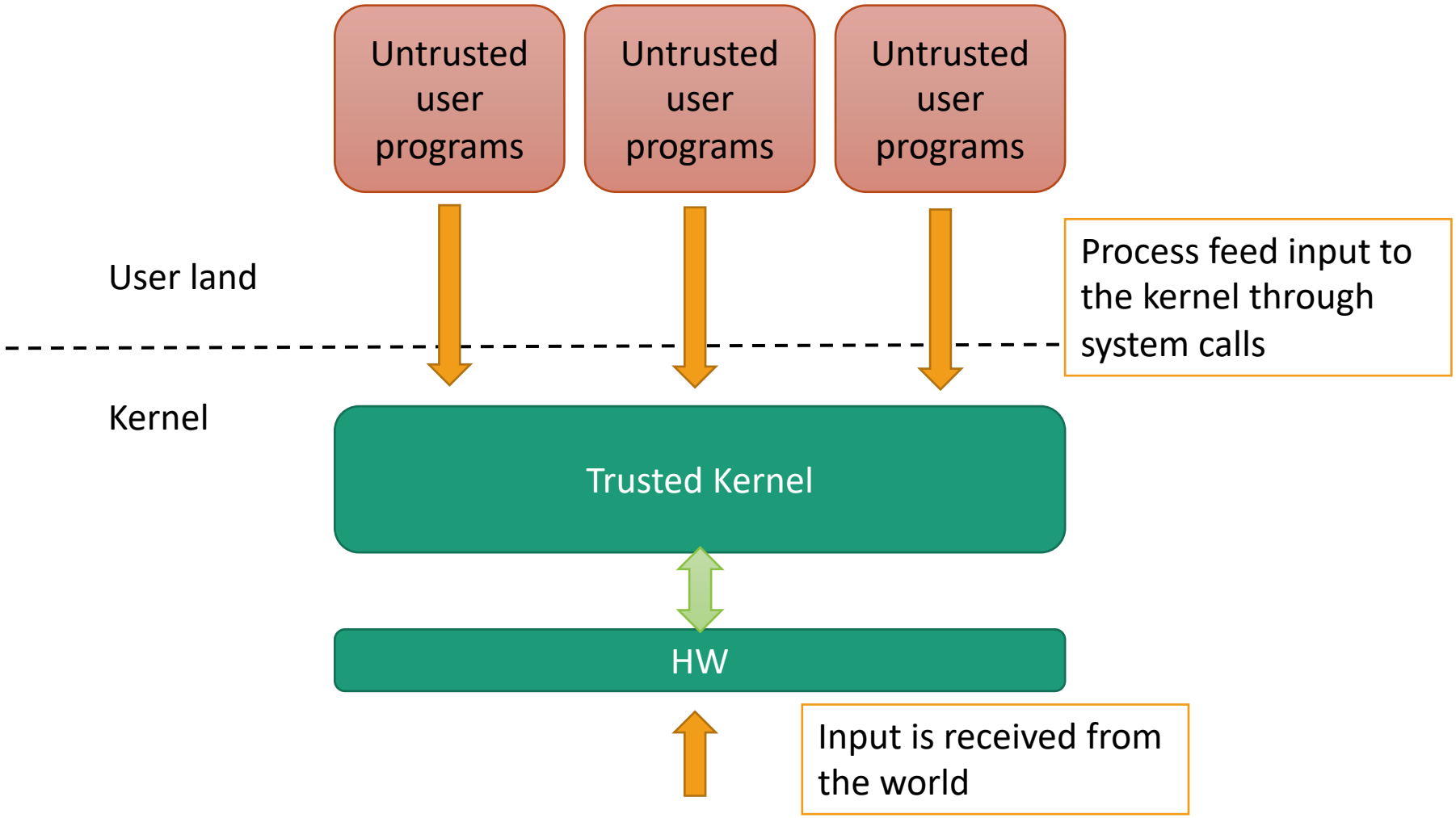


Kernel

What Triggers These Bugs?



What Triggers These Bugs?

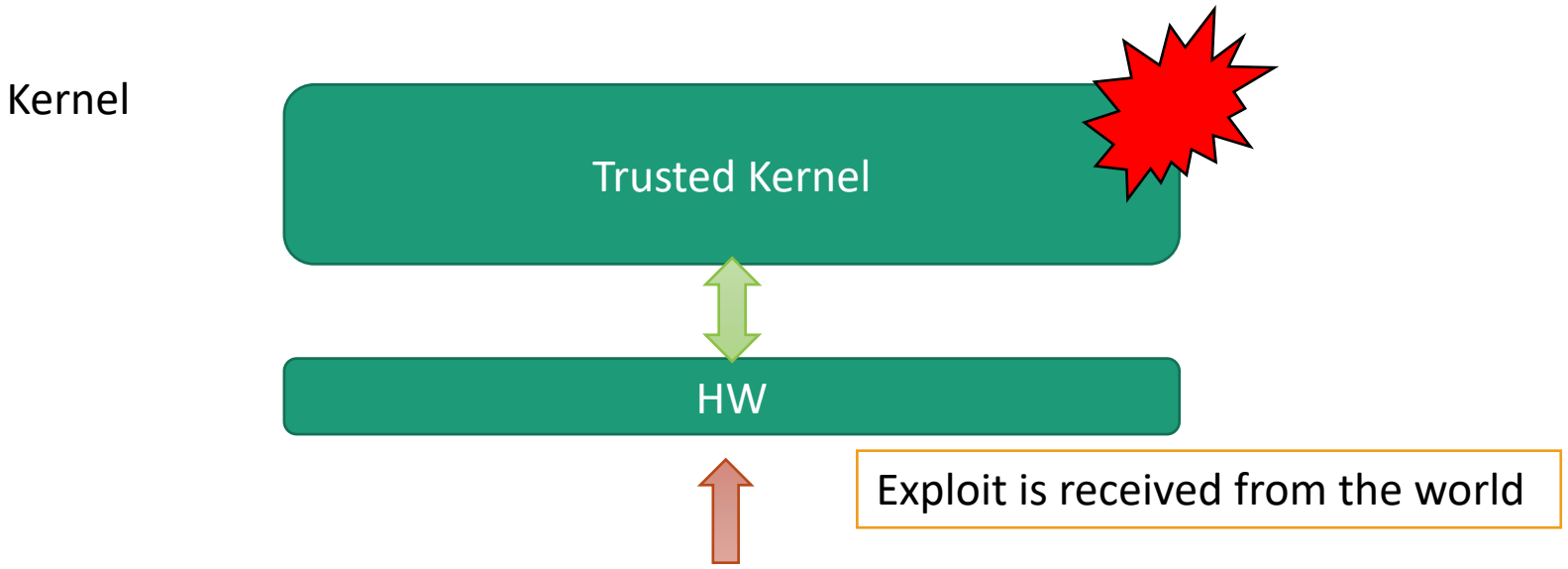


Remote Kernel Exploitation

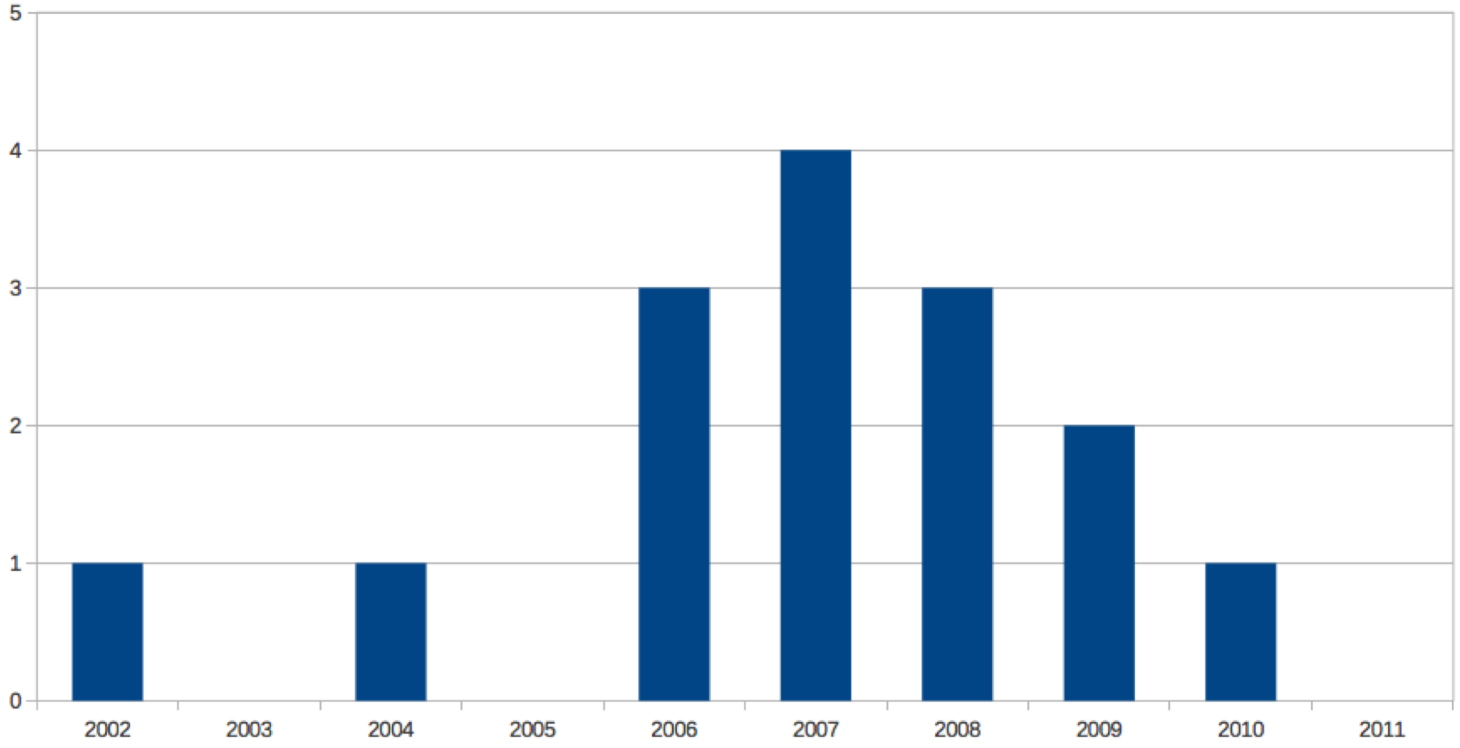
Extremely powerful attack, but rare and hard to pull off

Attacker has limited control of context

- Can only drive the kernel with external input
- Exploits may gain control in interrupt context
- When things go wrong the system crashes



Difficult but Still Possible



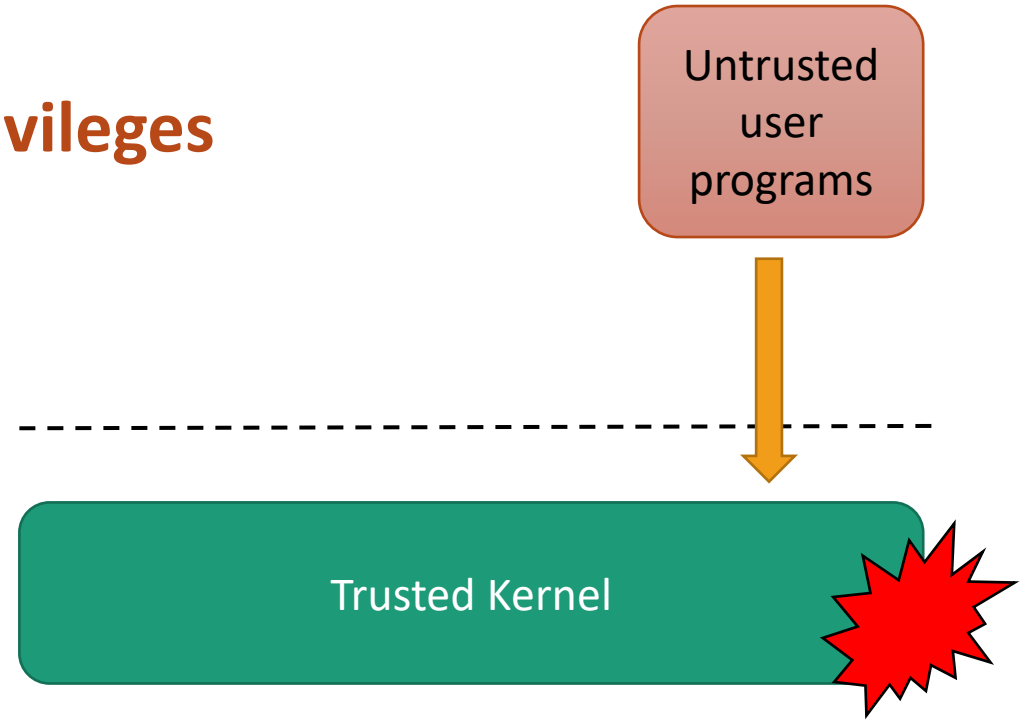
http://vulnfactory.org/research/h2hc-remote.pdf

Most Kernel Exploits

A malicious or compromised application performs the exploit against the kernel

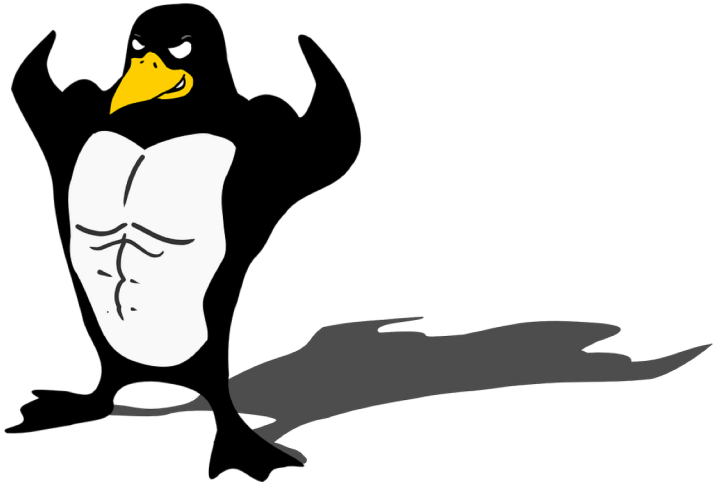
Usually, through supplying carefully crafted input to a system call

Attackers elevate their privileges



Kernel vs Applications

ASLR, W^X, heap & stack protection, CFI



Applications

KASLR, W^X, slab red zones (memory protection), SMEP, SMAP

Kernel

Enter Return-to-User Attacks

In simple words, gain control of the IP while executing in kernel space and redirect to execute code in user space

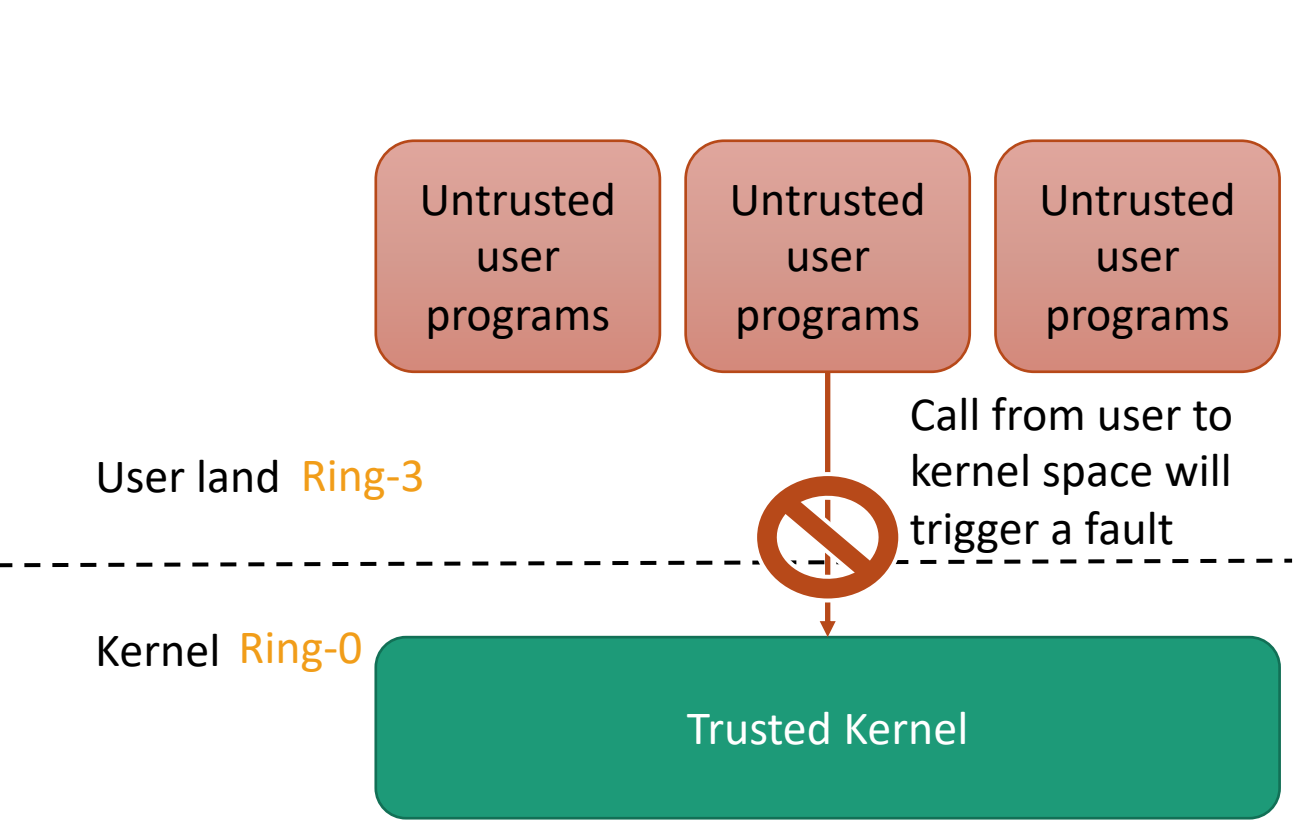
User space code is prepared by the attacker

- No restrictions essentially

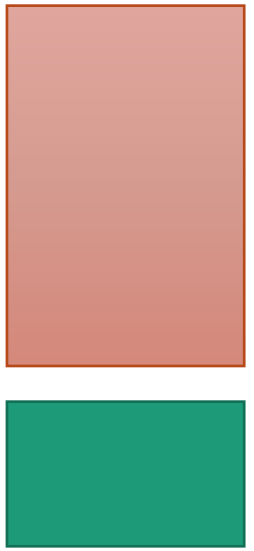
But code runs with kernel privileges

User-Kernel Separation

User-to-kernel boundary crossing protected by hardware

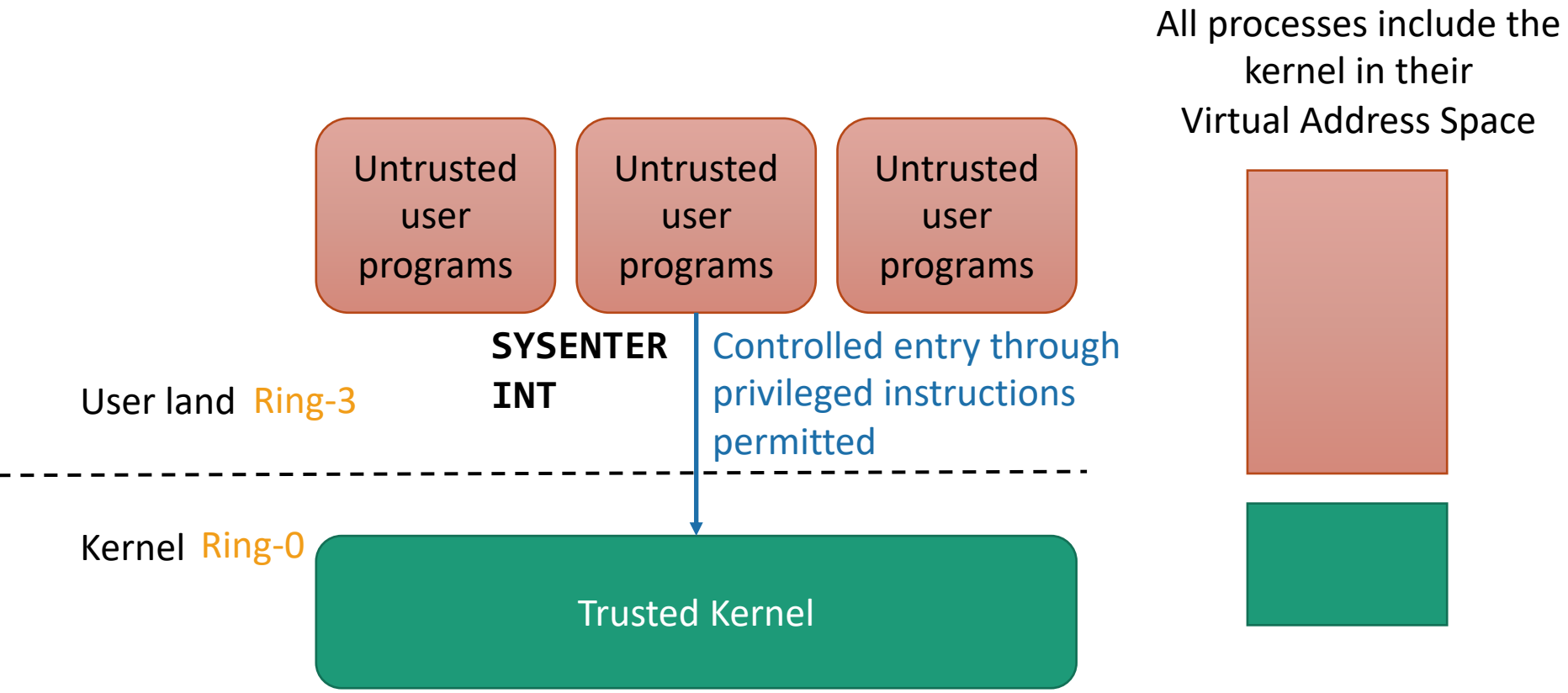


All processes include the kernel in their Virtual Address Space



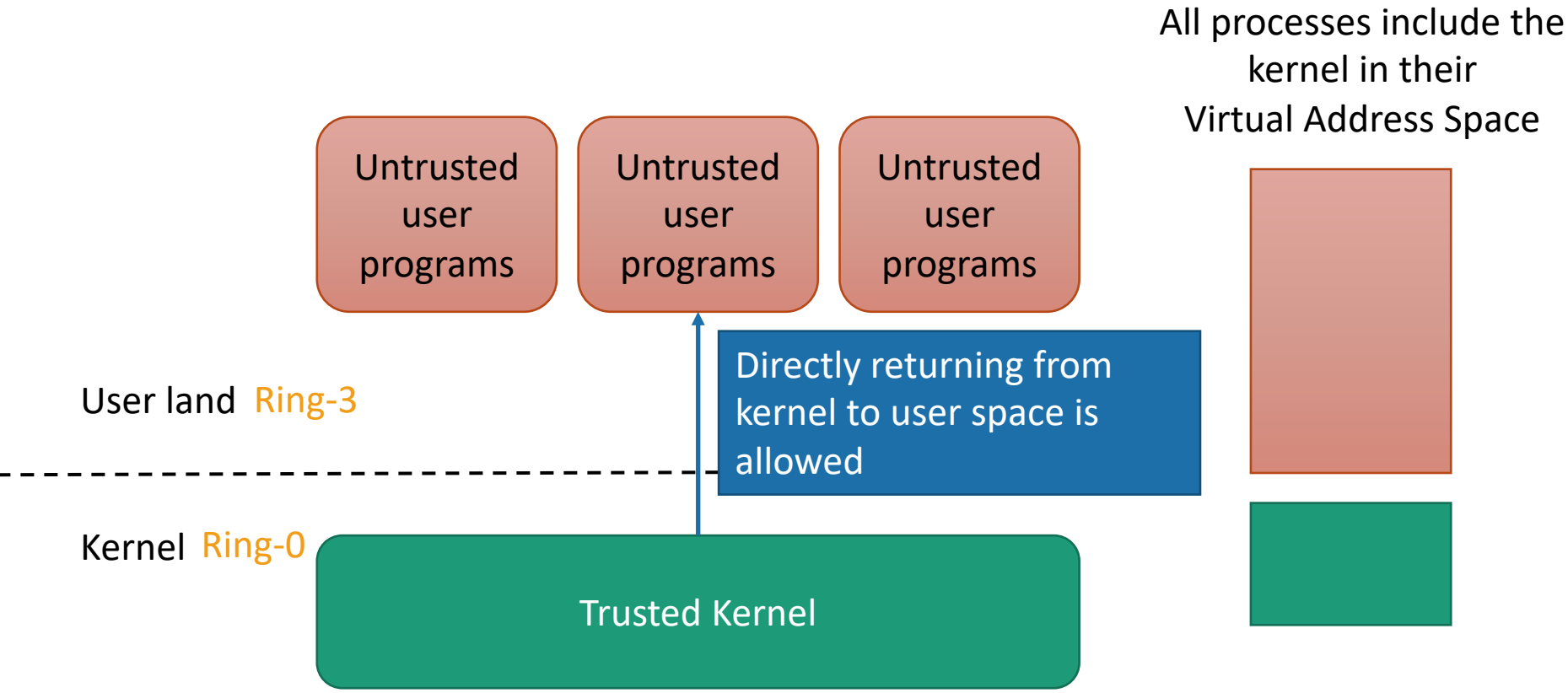
User-Kernel Separation

User-to-kernel boundary crossing protected by hardware



User-Kernel Separation

Kernel-to-user boundary not protected



Null-pointer Dereferences

In user space

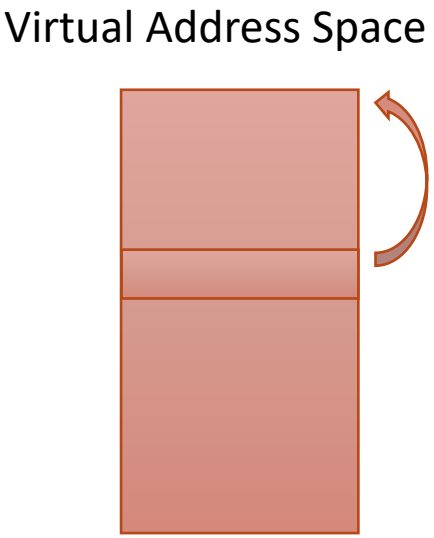
Only causes a memory fault

Application contains bug that nullifies a pointer

- p.my_callback(); = 0

Usually nothing is mapped at address 0

Application crashes



Null-pointer Dereferences

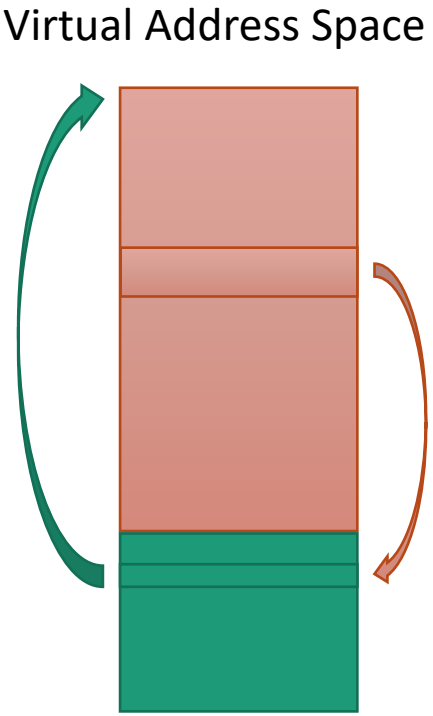
In kernel space

Application performs system call

Kernel contains bug that nullifies pointer

- p.my_callback(); = 0

System crashes because usually there is nothing mapped at address 0



Null-pointer Exploits

In kernel space

User places malicious shellcode at address 0

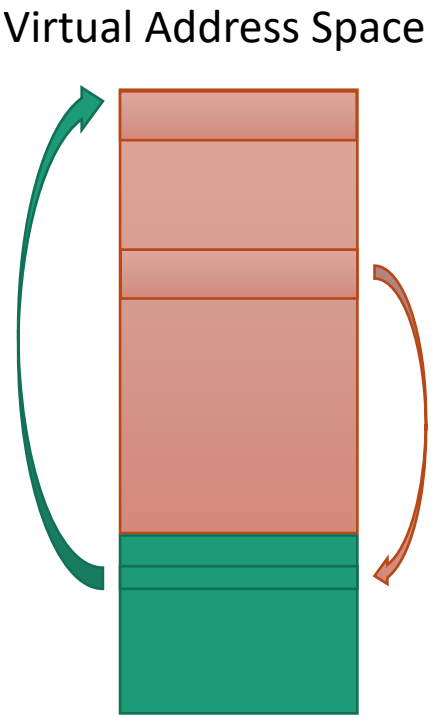
Application performs system call

Kernel contains bug that nullifies pointer

- p.my_callback(); = 0

User executes his own code as the kernel

- Privilege escalation attack



Solutions

SMEP/SMAP feature on newer processors

- Arbitrary transfers from kernel to user space prevented by HW
- Only special instruction permit return to user space

Bypasses have been demonstrated

- Use code reuse to disable SMEP/SMAP
- Through directly mapped memory
- Reading: https://www3.cs.stonybrook.edu/~mikepo/papers/ret2dir.sec14.pdf