# INF3510 Information Security
# Spring 2015

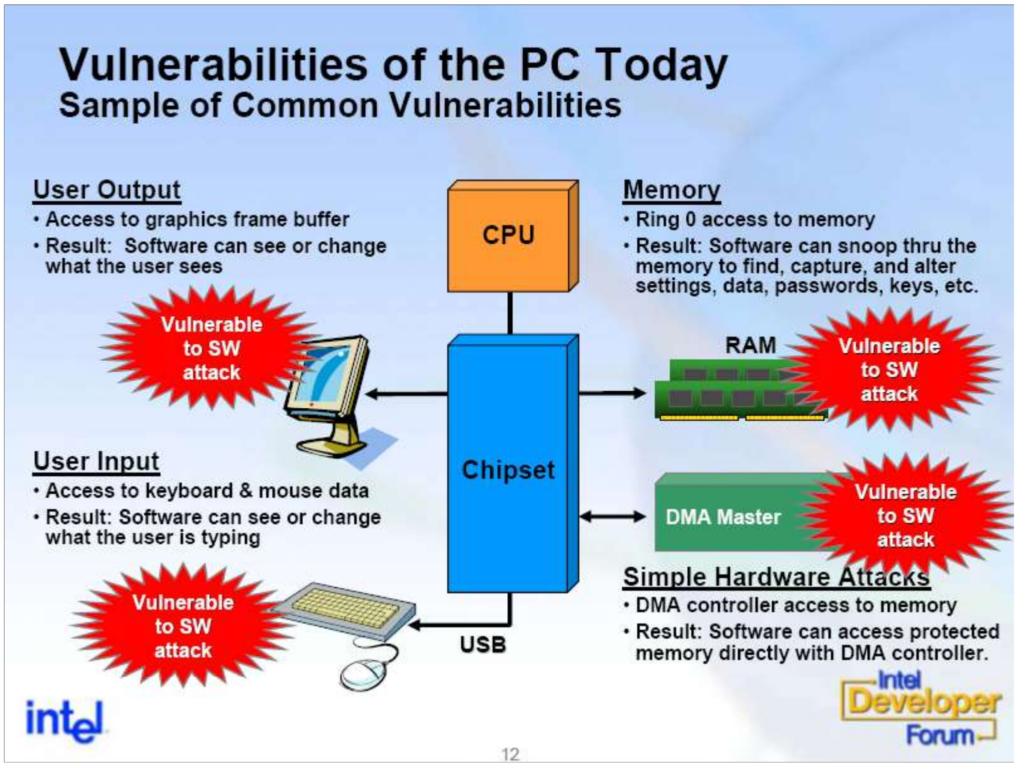
## Lecture 4
## Computer Security

University of Oslo
Audun Jøsang

## Lecture Overview

- Fundamental computer security concepts
- CPU and OS kernel security mechanisms
- Virtualization
- Memory Protection
- Trusted computing and TPM






## Meaningless transport defences when endpoints are insecure

"Using encryption on the Internet is the equivalent of arranging an armored car to deliver credit card information from someone living in a cardboard box to someone living on a park bench."

(Gene Spafford)

## Approaches to strengthening platform security

- Harden the operating system
  - SE (Security Enhanced) Linux, Trusted Solaris, Windows Vista/7/8
- Add security features to the CPU
  - Protection Layers, NoExecute, ASLR
- Virtualisation technology
  - Separates processes by separating virtual systems
- Trusted Computing
  - Add secure hardware to the commodity platform
  - E.g. TPM (Trusted Platform Module)
- Rely on secure hardware external to commodity platform
  - Smart cards
  - Hardware tokens



## TCB – Trusted Computing Base

- The trusted computing base (TCB) of a computer system is the set of all hardware, firmware, and/or software components that are critical to its security, in the sense that bugs or vulnerabilities occurring inside the TCB might jeopardize the security properties of the entire system.
- By contrast, parts of a computer system outside the TCB must not be able to breach the security policy and may not get any more privileges than are granted to them in accordance to the security policy

(TCSEC – Trusted Computer Evaluation Criteria, 1985).



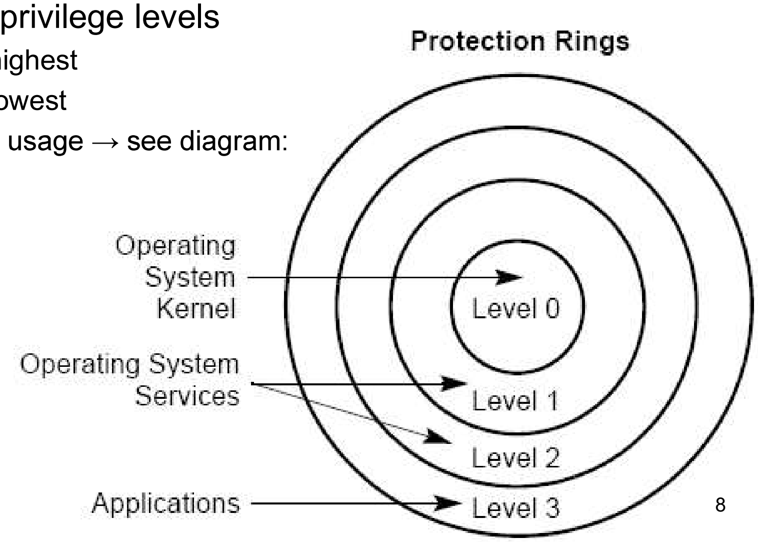
## Reference Monitor

- Reference monitor is the security model for enforcing an access control policy over subjects' (e.g., processes and users) ability to perform operations (e.g., read and write) on objects (e.g., files and sockets) on a system.
  - The reference monitor must always be invoked (complete mediation).
  - The reference monitor must be tamperproof (tamperproof).
  - The reference monitor must be small enough to be subject to analysis and tests, the completeness of which can be assured (verifiable).
- The security kernel of an OS is a low-level (close to the hardware) implememtation of a reference monitor.



## OS security kernel as reference monitor

- Hierarchic security levels were introduced in X86 CPU architecture in 1985 (Intel 80386)
- 4 ordered privilege levels
  - Ring 0: highest
  - Ring 3: lowest
  - Intended usage → see diagram:



# What happened to rings 1 & 2 ?

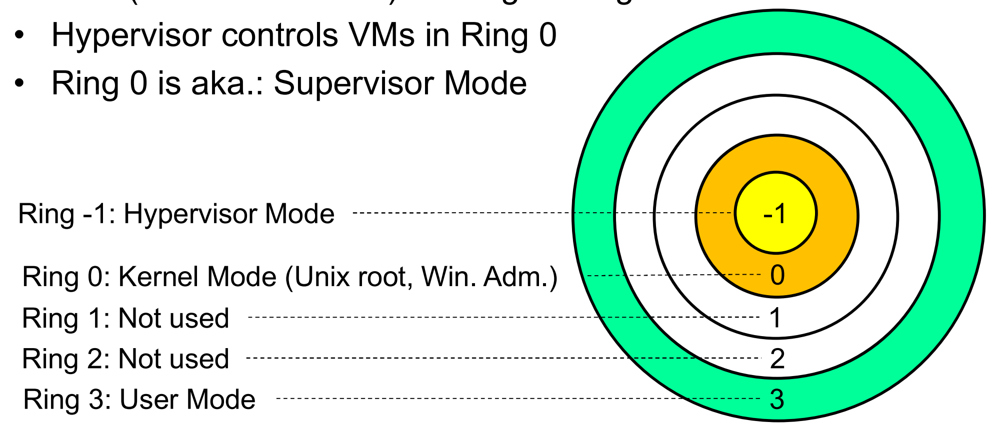
... it eventually became clear that the hierarchical protection that rings provided did not closely match the requirements of the system programmer and gave little or no improvement on the simple system of having two modes only. Rings of protection lent themselves to efficient implementation in hardware, but there was little else to be said for them. [...]. This again proved a blind alley...

Maurice Wilkes (1994)



# CPU Protection Ring structure from 2006

- New Ring -1 introduced for virtualization.
- Necessary for protecting hypervisor from VMs (Virtual Machines) running in Ring 0.
- Hypervisor controls VMs in Ring 0
- Ring 0 is aka.: Supervisor Mode





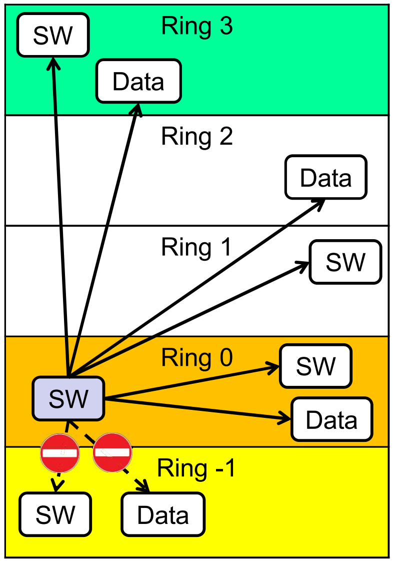

# Privileged Instructions

- Some of the system instructions (called "privileged instructions") are protected from use by application programs.
- The privileged instructions control system functions (such as the loading of system registers). They can be executed only when the Privilege Level is 0 or -1 (most privileged).
- If one of these instructions is attempted when the Privilege Level is not 0 or -1, then a general-protection exception (#GP) is generated, and the program crashes.



# Principle of protection ring model

- A process can access and modify any data and software at the same or less privileged level as itself.
- A process that runs in kernel mode (Ring 0) can access data and SW in Rings 0, 1, 2 and 3
  - but not in Ring -1
- The goal of attackers is to get access to kernel or hypervisor mode.
  - through exploits
  - by tricking users to install software



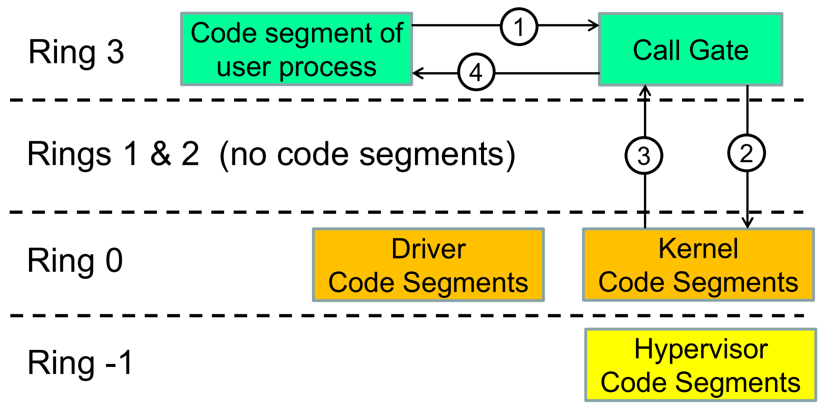
# User processes access to system resources

- User processes need to access system resources (memory and drivers)
- User application processes should not access system memory directly, because they could corrupt memory.
- The CPU must restrict direct access to memory segments and other resources depending on the privilege level.

- Question 1: How can a user process execute instructions that require kernel mode, e.g. for writing to memory ?
  - Answer: The CPU must switch between privilege levels
- Question 2: How should privilege levels be switched?
  - Answer: Through Controlled invocation of code segments



# Controlled Invocation of code segments






# Controlled Invocation

- The user process executes code in specific code segments.
- Each code segment has an associated mode which dictates the privilege level the code executes under.
- Simply setting the mode of user process code to Kernel would give kernel-privilege to user process without any control of what the process actually does. Bad idea!
- Instead, the CPU allows the user process to call kernel code segments that only execute a predefined set of instructions in kernel mode, and then returns control back to the user-process code segment in user mode.
- We refer to this mechanism as controlled invocation.



# Platform Virtualization

# Virtual machines (VM)

- A software implementation of a machine (OS) that executes programs like a real machine (traditional OS)
- Example:
- Java Virtual Machine (JVM)
  - JVM accepts a form of computer intermediate language commonly referred to as Jave bytecode.
    - "compile once, run anywhere“
  - The JVM translates the bytecode to executable code on the fly
- Platform Virtualization
  - Simultaneous execution of multiple OSs on a single computer hardware, so each OS becomes a virtual computing platform

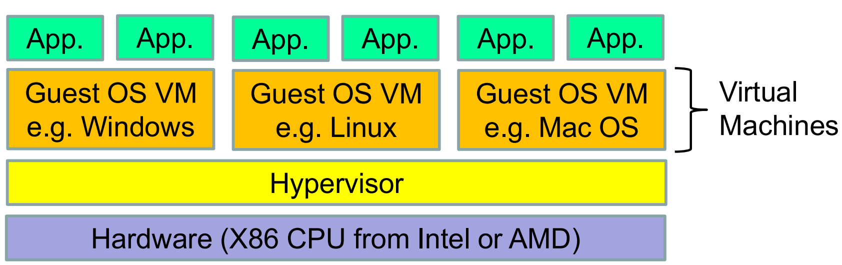


# Platform Virtualization

- Hypervisor (aka. VMM - Virtual Machine Monitor) is needed to manage multiple guest OSs (virtual machines) in the same hardware platform.
- Many types of hypervisors available
  - VMWare is most known Commercial product
    - Free version comes with a limitations
  - VirtualBox is a hypervisor for x86 virtualization
    - It is freely availably under GPL
    - Runs on Windows, Linux, OS X and Solaris hosts
  - Hyper-V is Microsoft’s hypervisor technology
    - Requires Windows Server



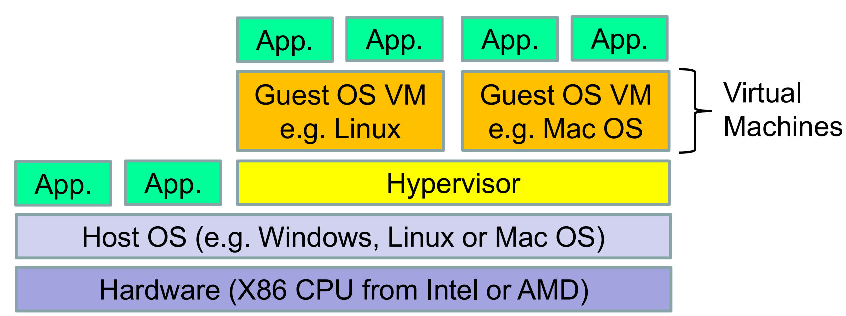
# Type 1 VM Architecture (advanced virtualization)




- No host OS
- Hypervisor runs directly on hardware
- High performance
- Traditionally limited GUI, but is improved in modern versions
- HW support can be an issue

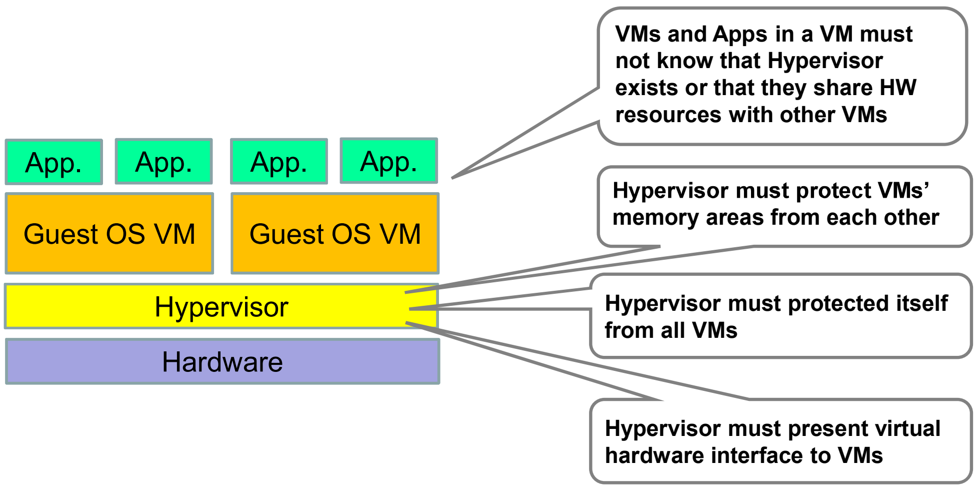


# Type 2 VM Architecture (simple virtualization)



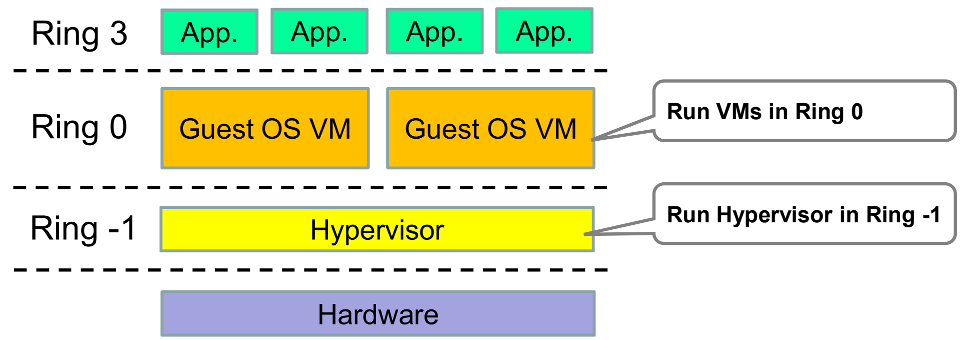

- Hypervisor runs on top of host OS
- Performance penalty, because hardware access goes through 2 OSs
- Traditionally good GUI
- Traditionally good HW support, because host OS drivers available

# Challenges of Running VMs





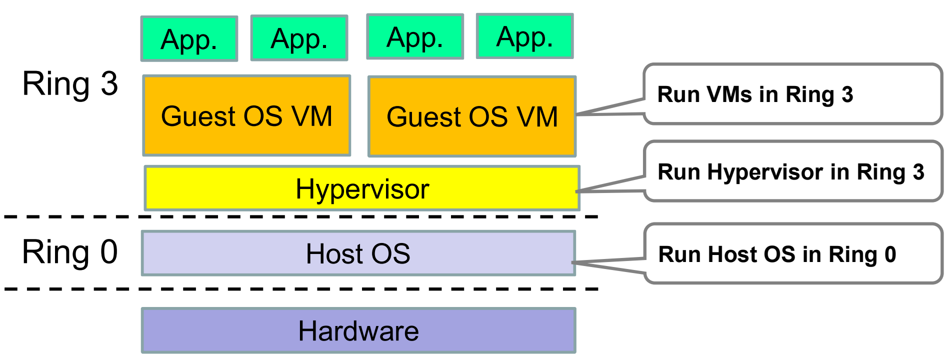

# Type 1 VM Architecture Ring Allocation




- Guest OS VMs are less privileged than the hypervisor.
- Hypervisor is well protected from the VMs.
- Good security !



# Type 2 VM Architecture Ring Allocation



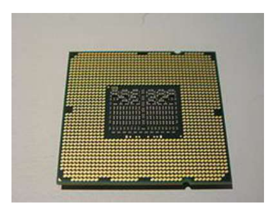

- Guest OS VMs run in Ring 3.
- Guest OS VMs call privileged instructions that are forbidden in Ring 3.
- Forbidden instructions cause exceptions that are handled by interrupt/exception handler to be executed.
- Slow performance !



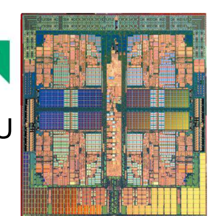
# Hardware support for virtualization

- Modern Intel and AMD X86 CPUs support virtualization
  - Intel-VT (Intel Virtualization Technology)
  - AMD-V (AMD Virtualization)
- Must be enabled in BIOS
  - Can be enabled and disabled
  - Computers with single OS typically have virtualization disabled
- Access to data- and code segments for hypervisor can be restricted to processes running in hypervisor mode
- Some instructions are reserved for hypervisor mode

Intel Core i7 CPU



AMD Phenom CPU



# Why use platform virtualization

- Efficient use of hardware and resources
  - Improved management and resource utilization
  - Saves energy
- Improved security
  - Malware can only infect the VM
  - Safe testing and analysis of malware
  - Isolates VMs from each other
- Distributed applications bundled with OS
  - Allows optimal combination of OS and application
  - Ideal for cloud services
- Powerful debugging
  - Snapshot of the current state of the OS
  - Step through program and OS execution
  - Reset system state



# Hypervisor examples of use

Google data center

- Cloud providers run large server parks
  - Each customer gets its own VM
  - Many customers share the same hardware
  - Migrated VMs between servers to increase/reduce capacity
- Testing and software analysis
  - Potentially damaging experiments can be executed in isolated environment
  - Take a snapshot of the current state of the OS
  - Use this later on to reset the system to that state
  - Malware Analysis




# Memory Protection

# Buffer overflow

- A program tries to store more data in a buffer than it was intended to hold.
- Example:
  - Assume a 5 bytes buffer to store a variable in memory:

| | | | | |
|---|---|---|---|---|
| | | | | |

  - Write10 bytes to buffer, then 5 extra bytes get overwritten

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| a | b | c | d | e | f | g | h | i | j |

  - If the overwritten part contained a return pointer or software, it is possible for the attacker to execute his own instructions.
- Many attacks are based on buffer overflow techniques

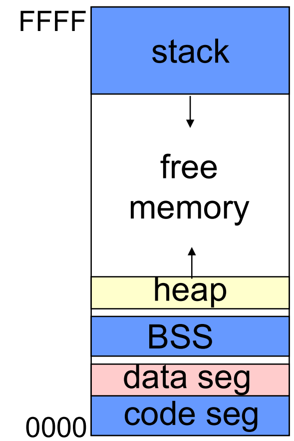
# Buffer Overflow

- Buffer overflow is when written data size > buffer size
  - Results in neighbouring buffers being overwritten
- Unintentional buffer overflow crashes software, and results in unreliability software.
- Intentional buffer overflow is when an attacker modifies specific data in memory to execute malware
- Attackers target return addresses (specify the next piece of code to be executed) and security settings.
- In languages like C or C++ the programmer allocates and de-allocates memory.
- Type-safe languages like Java guarantee that memory management is 'error-free'.



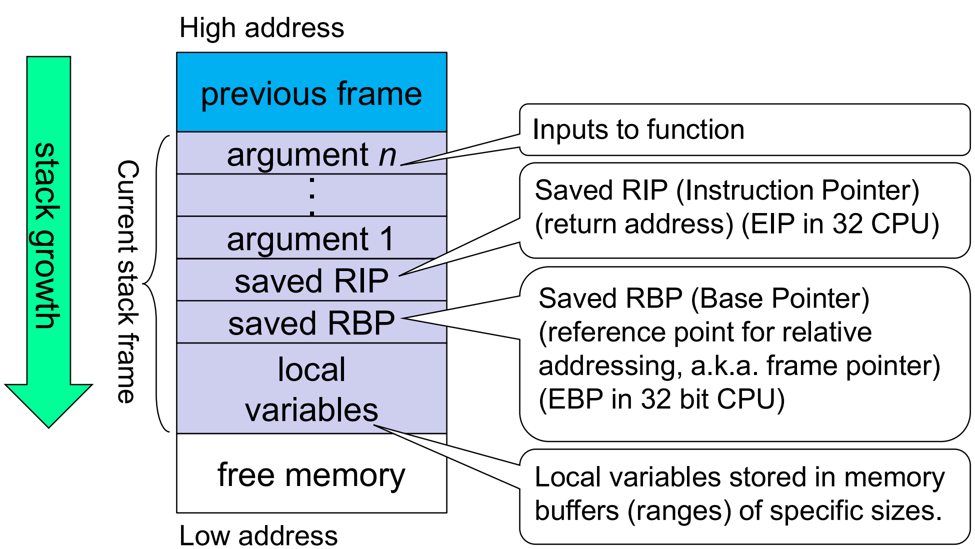
# Memory corruption and buffer overflow

- The stack contains memory buffers that hold return address, local variables and function arguments. It is possible to decide in advance where a particular buffer will be placed on the stack.
- Heap: dynamically allocated memory; more difficult but not impossible to decide in advance where a particular buffer will be placed on the heap.
- BSS: Block Segment of Static Variables





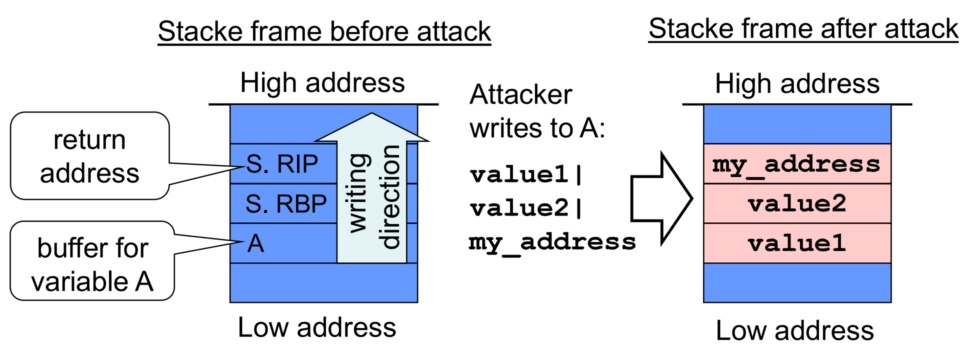

# Stack Frame – Layout






# Stack-based Overflows

- Find a buffer on the runtime stack that can overflow.
- Overwrite the return address with the start address of the code you want to execute.
- The code can also be injected by overflowing buffers.
- You can now execute your own code.

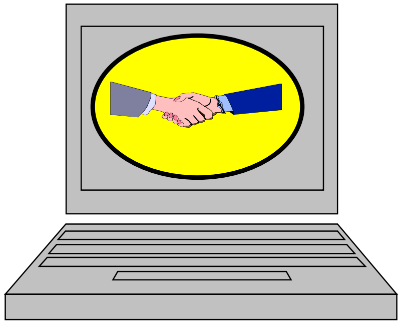


## Defences against memory corruption

- Hardware mechanisms
  - NX (No eXecute) bit/flag in stack memory
    - Injected attacker code will not execute on stack
- OS / compiler mechanisms
  - Stack cookies: detects corruption at runtime
  - ASLR (Address Space Layout Randomization)
    - Makes it difficult to locate functions in memory
- Programming language
  - Type safe languages like Java and C#
- Programming rules
  - Avoid vulnerable functions like
    - strcpy (use strncpy instead)
    - gets (use fgets instead)



## Trusted Computing





## Trusted Computing Motivation

- Software alone can not be trusted.
- Malware infection in OS kernel remains undetected by anti-malware tools.
- Physical access to computers opens up for attacks that can circumvent traditional TCBs (Trusted Computing Base), e.g. secure operating systems.
- Remote parties do not know the status of systems they are communicating with.
- Remote parties do not know the physical identity of hosts they are communicating with.



## Basic idea of Trusted Computing

- Use specialised **security hardware** as part of TCB in a computer system
  - Can not be compromised by malware
  - Can verify the integrity of OS kernel
  - Can make physical tampering difficult
  - Can report status of system to remote parties
  - Can report identity of system to remote parties
- Gives increased level of trust that the system will perform as expected/specified

# Need for trusted hardware

- Computing platforms are typically deployed in hostile environments, in contrast to 1960's and 1970's protected computing centres
  - Users are considered to be hostile
- Trusted Computing is not a new idea!
- Tygar and Yee: A System for Using Physically Secure Coprocessors 1991
  - Cryptography assumes the secrecy of keys
  - Secrecy requires physical security

    (J. D. Tygar and B. Yee. A System for Using Physically Secure Coprocessors, *Technical Report CMU-CS-91-140R*, Carnegie Mellon University, May 1991)



# What is "trust" in the sense of TC?

- To have confidence in assumptions about security
- Trust is to believe that security assertions will hold

  "*A trusted component, operation, or process is one whose behaviour is assumed to be correct under any operating condition, and which is assumed to resist subversion by malicious software, viruses, and manipulations*"
- A trusted component enforces the security policy as long as these assumptions hold
- A trusted component violates the security policy if it breaks
- Q1: How do you know that a component is 'trustworthy', i.e. that it will not break ?
- Q2: Trusted by whom to do what ?
  - Trusted by **user**, by **vendor**, or by **3rd party (NSA)**
  - What if they have conflicting interests ?



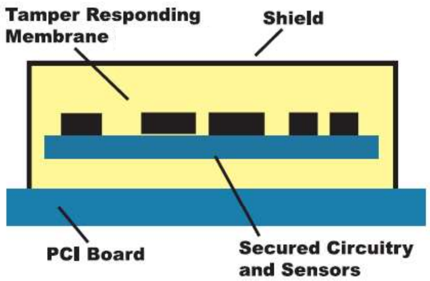
# Characteristics of Trusted Hardware

- Physically secure hardware component
  - Assumed not to break because it's hardware
- Environmental monitoring (temperature, power supply, structural integrity)
- Tamper responsive
- Implementations
  - CPU
  - ROM for OS and application code
  - Specialized hardware for cryptography and for storing secrets



# Trusted Hardware – Example

- IBM 4765 Secure Coprocessor



# Trusted Computing Group (TCG)

•TCG Promoters

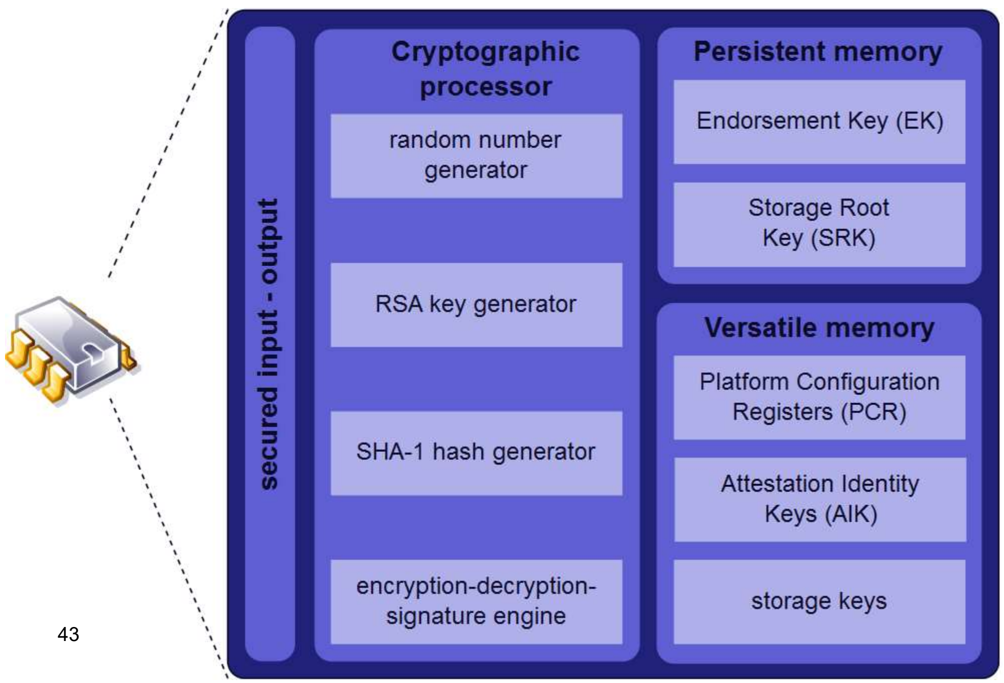


# TCG History & Evolution

- October 1999: TCPA formed
  - Trusted Computing Platform Alliance
  - Founders: IBM, HP, Compaq, Intel and Microsoft
- 2001: 1st TPM specification released
  - Trusted Platform Module
- 2002: TCPA changes its name to TCG
  - Trusted Computing Group
  - Industry standards organization
- 2003: TCPA TPM spec. adopted by TCG as TPM 1.2
- 2011: Latest TPM spec. 1.2 Ver.116
- 2012: Draft TPM Specification 2.0 published
  - TPM 2.0 spec. not compatible with TPM 1.2 spec.
- 2015: Official TPM specification 2.0



# TPM Functionality






# TPM usage

- TPM is both the name of a standard and a chip
- TPM chip at the heart of hardware / software approach to trusted computing
- Current TPM chips implement TPM spec. 1.2
  - Latest version of TPM spec. 1.2 is from 2011
- TPM chip mounted on motherboard,
- TPM equipped computing platforms
  - Laptops, servers, pads, mobile phones
- Used by software platforms
  - Windows Vista / 7 / 8, Linux, and MAC OS
- Supports 3 basic services:
  - Authenticated/measured boot,
  - Sealed Storage / Encryption
  - Remote attestation,

# Two modes of booting

- Secure boot with UEFI (not with TPM, see UEFI later)
  - The platform owner can define expected (trusted) measurements (hash values) of OS software modules.
  - Hash values stored in memory signed by private PK (Platform Key).
  - Public PK stored in secure firmware on platform
  - Measured has values can be compared with stored values.
  - Matching measurement values guarantee the integrity of the corresponding software modules.
  - Boot process terminates if a measurement does not match the stored value for that stage of the boot process.
- Authenticated/Measured boot with TPM
  - Only records measured values in PCRs and reports to remote party
  - Does not terminate boot of measured values are not as expected



# Sealed Storage / Encryption

- Encrypts data so it can be decrypted
  - by a certain machine in given configuration
- Depends on
  - Storage Root Key (SRK) unique to machine
  - Decryption only possible on unique machine
- Can also extend this scheme upward
  - create application key for desired application version running on desired system version
- Supports disk encryption



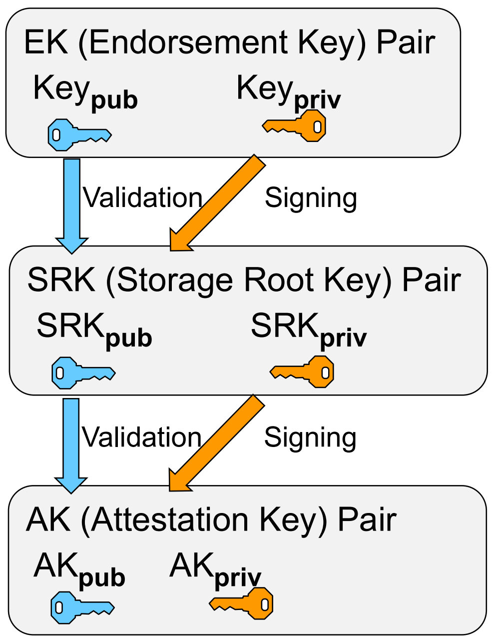
# Remote Attestation

- TPM can certify configuration to others
  - with a digital signature in configuration info
  - giving another user confidence in it
  - Based on Attestation Key (AK)
- Remote parties can validate signature based on a PKI
- Provides hierarchical certification approach
  - trust TPM, then trust the OS, then trust applications



# TPM Key Hierarchy

- Endorsement Key (EK)
  - Created once
  - Stored securely in non-volatile memory
- Storage Root Key (SRK)
  - Stored security in non-volatole memory
  - Validated by EK
- Attestation Key (AK)
  - Used for remote attestation
  - Validated by SRK
- Custom keys
  - Possible to create additional keys and validate that they are created under SRK.



# UEFI
# Unified Extensible Firmware Interface

– What is UEFI?
  - Replaces traditional BIOS
  - Like BIOS it hands control of the pre-boot environment to an OS
– Key Security Benefits:
  - Secure Boot



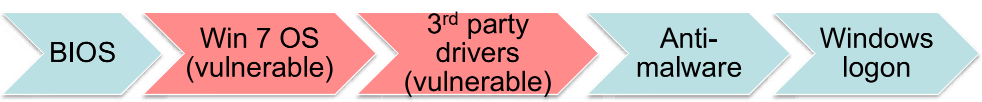
# UEFI Secure Boot

- Prevents loading unsigned drivers or OS loaders
- When secure boot is enabled, it is initially placed in "setup" mode, which writes public key known as the "Platform key" (PK) to firmware.
- Once the key is written, secure boot enters "User" mode, where only drivers and loaders signed with the platform key can be loaded by the firmware.
- "Key Exchange Keys" (KEK), signed by private PK, can be added to a database stored in memory to allow other signatures by other than PK.
- Secure boot supported by Win 8, Win Server 2012, Fedora, OpenSuse, and Ubuntu
- Does not require TPM



# TPM Secure Boot in Windows 8

**•Windows 7 boot**




- Malware is able to start before Windows 7 and Anti-malware
  - Malware able to hide and remain undetected
  - Systems can be completely compromised

**•Windows 8 boot**


- Possible to start anti-malware early in the Windows 8 boot process
  - Early Launch Anti-Malware (ELAM) driver is signed by Microsoft
  - Malware can no longer bypass Anti-Malware inspection
  - UEFI (Unified Extensible Firmware Interface) replaces BIOS



## End of lecture