Virtualization -, Container-, and a bit of Serverless Security

Pék Gábor

CrySyS Lab, BME HIT

pek@crysys.hu

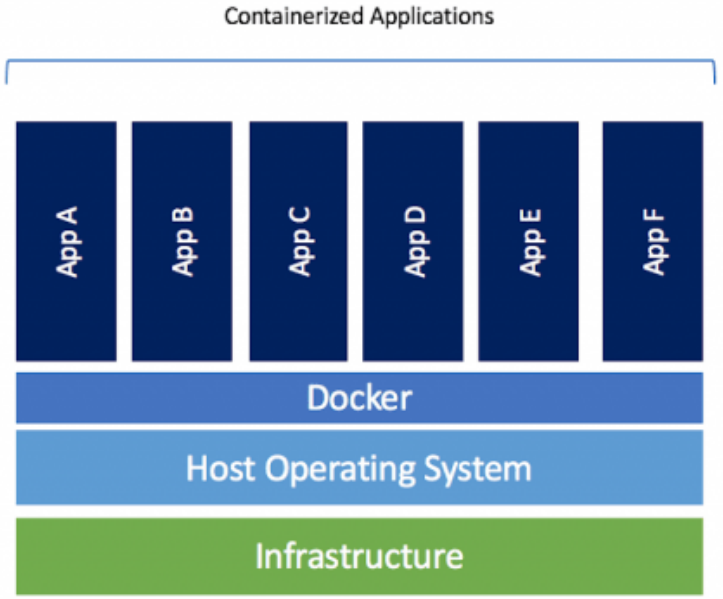
Outline

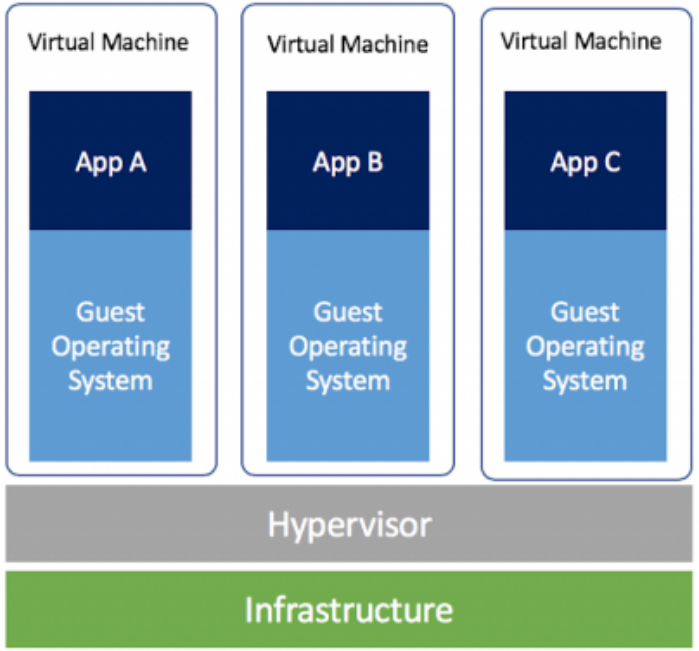
- Virtualization security
  - Detecting virtualization
  - Hardware virtualization escapes
  - I/O Virtulization attacks
  - Protecting hardware virtualization
- Container security
  - Containerization History
  - Docker engine security
  - Container vulnerability scanning
- Serverless security
  - Quick technology overview

Quick overview of virtualization approaches

- Virtual machines (in case of System Virtualization)
  - Full-fledged environment with emulated or hardware virtualized resources
  - Not optimal for lightweight scaling
  - Hosted and Native
- Containers
  - Isolation with security countermeasures
  - More lightweight than VMs
- Serverless
  - Ideal for ephemeral tasks (e.g., function calls)

Containers vs. Virtual Machines





VIRTUALIZATION SECURITY

Virtualization – Introduction

- Virtualization is the creation of a *software-based (virtual)* representation of something
  - Hardware virtualization
  - Network virtualization
  - Storage virtualization
  - Application virtualization
  - Desktop virtualization
  - User state virtualization
  - And more…

Virtualization – Introduction

- Several benefits
  - Resource usage (and cost) optimization
  - Faster deployment
  - Flexibility
  - Convenience
  - High availability

- But what about security?

General issues

- Complexity issues
  - Virtualized systems are more complex
  - More software -> more code -> more bugs -> more vulnerabilities
  - More updates to install (patch management)
- Management difficulties
  - Who manages what?
  - It becomes easier to violate the *separation of duties* principle as systems become more intertwined
- The security of the core environment becomes more important
  - Breach = bigger damage, bigger outage

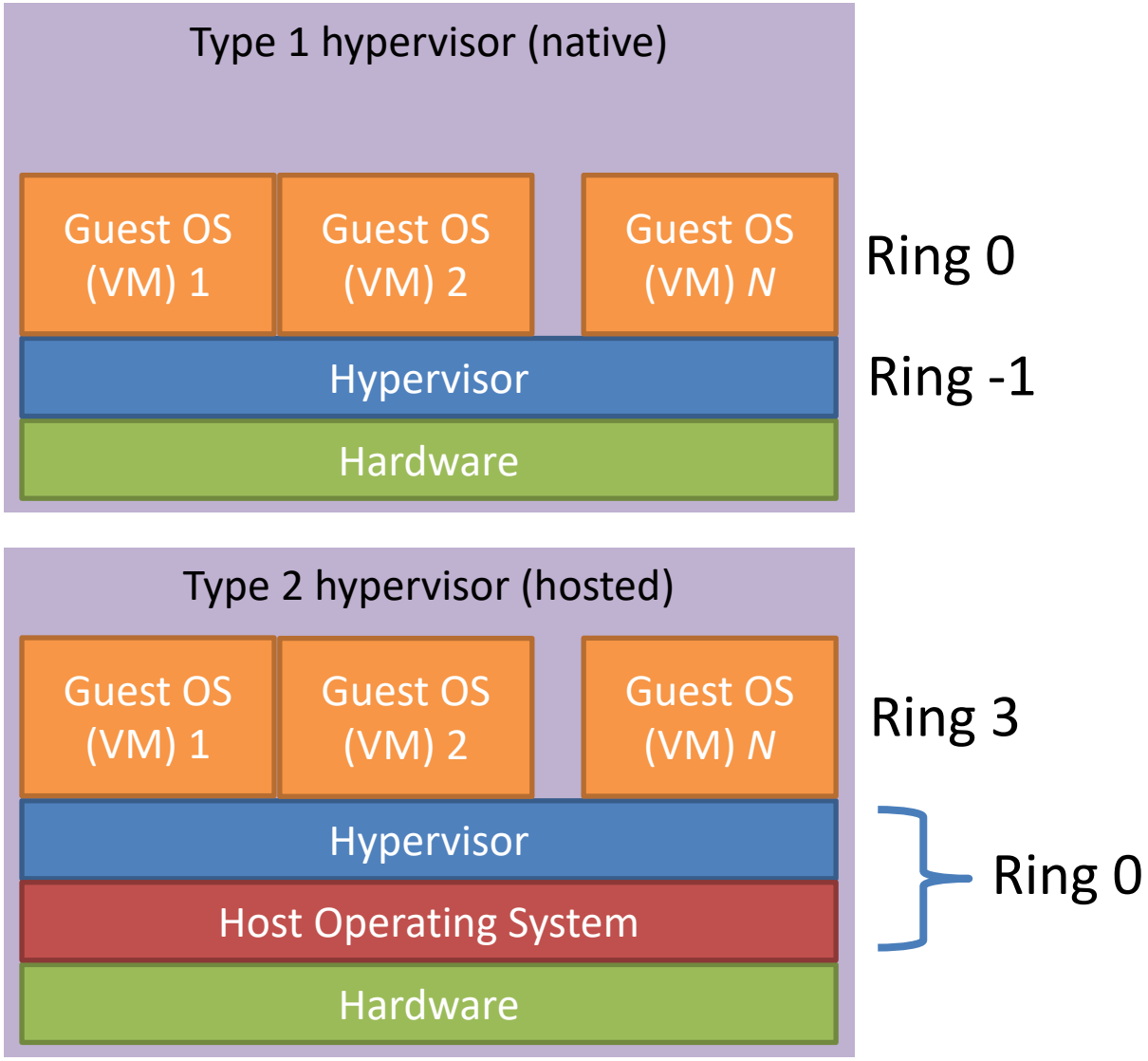
HARDWARE VIRTUALIZATION SECURITY

Hardware virtualization – Definitions

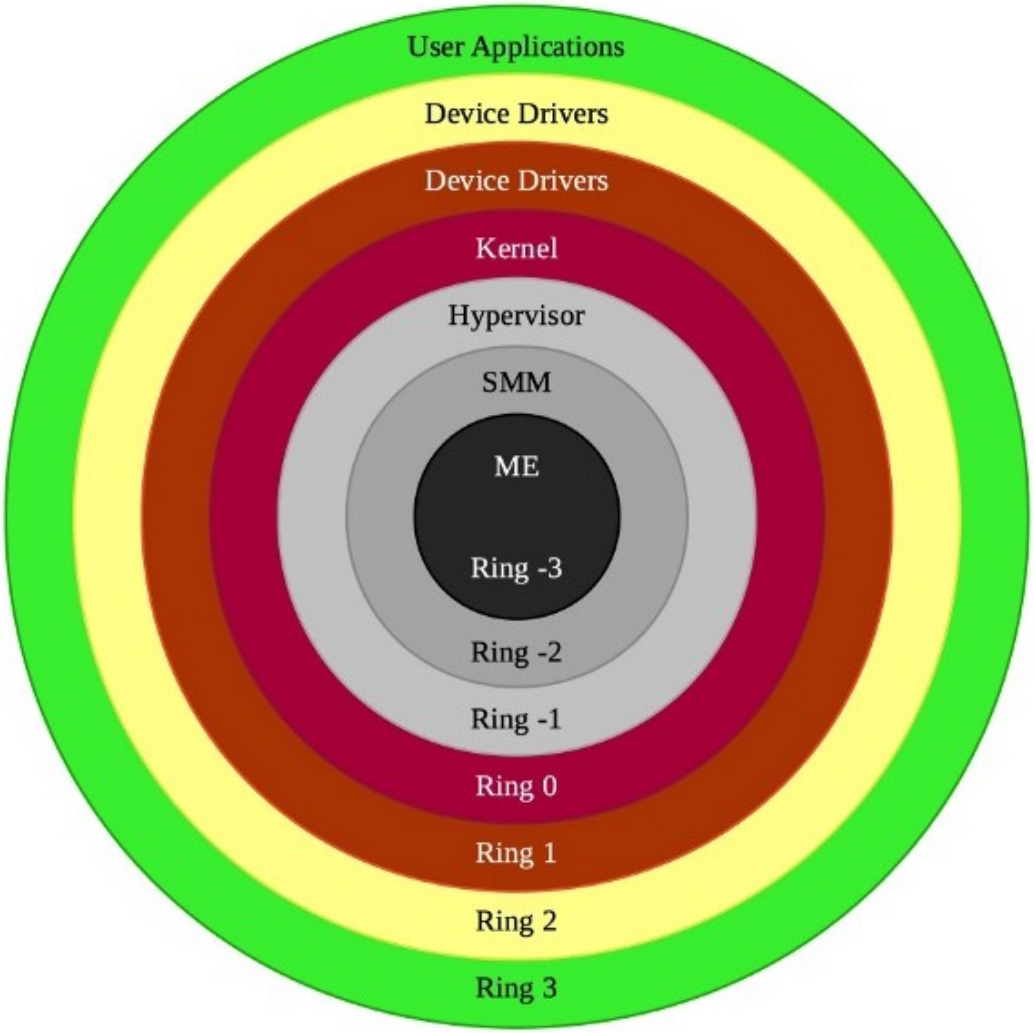
- Hypervisor (~ Virtual Machine Monitor, VMM): a piece of software that creates, manages, and runs virtual machines on top of some hardware
  - Schedules access to physical resources
  - Enforces security boundaries

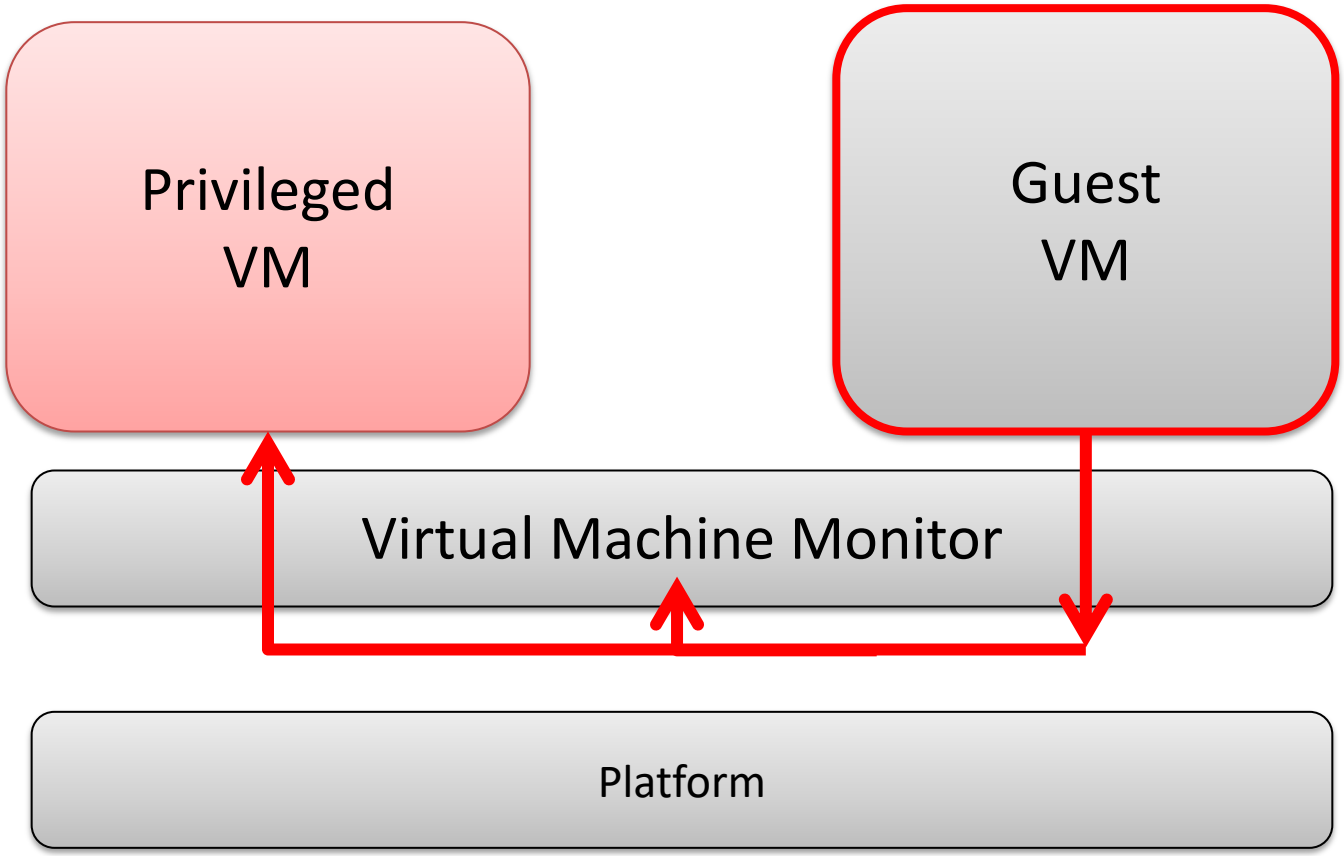
- Host: the physical machine the hypervisor is running on
- Guest (OS): a virtual machine and its virtual operating system

Hardware virtualization – Hypervisors



Rings


User Applications
Device Drivers
Device Drivers
Kernel
Hypervisor
SMM
ME
Ring -3
Ring -2
Ring -1
Ring 0
Ring 1
Ring 2
Ring 3



ATTACKING HARDWARE VIRTUALIZATION

Motivation for Hardware Virtualization Attacks

- Hardware virtualization is widely used today
  - Private and public clouds
- Security issues are not well understood
  - Yet another layer on the software stack
  - Lack of formal methods
  - Large code base

Challenges

- Requires deep understanding of
  - Virtualization concepts (cpu, i/o and memory virtualization)
  - Virtual Machine Monitors
  - Operating systems
- Goal: Implementing new attacks
  - That work on contemporary hardware and software
  - Call the attention of vendors for the weaknesses of current platforms

Hardware virtualization – Attacks

- By the level of interaction
  - Passive
    - » Hypervisor detection
    - » Network/environment monitoring
  - Active
    - » DoS
    - » VM escape
    - » Hyperjacking
- By direction
  - Guest-to-guest
  - Guest-to-host

Hypervisor detection

- Goal: to detect whether the program is running in a hypervisor

- Why?
  - Providing better compatibility/performance
  - Analysis evasion (e.g., malware)
  - Enforcing licensing restrictions

- How?
  - Via APIs
  - Environment-based detection
  - Hardware-based detection
  - Side-channel attacks

Hypervisor detection – Via APIs

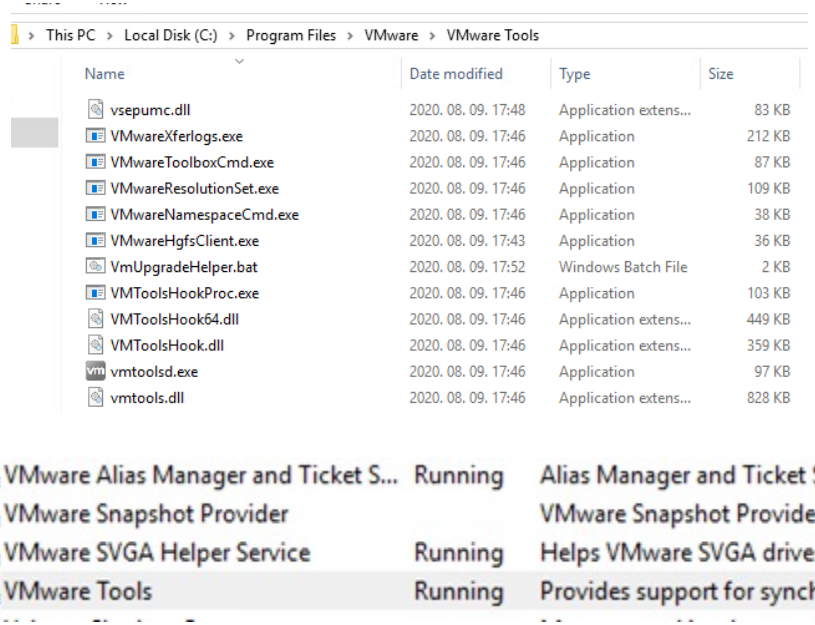
- Hypervisors generally don't care if their presence is detectable

- Some OS APIs yield different results when a Hypervisor is running
  - The exact purpose of some APIs is to tell the querier that they are running in a virtual machine
    - » E.g., HviIsAnyHypervisorPresent (Microsoft Windows)

Hypervisor detection – SW Environment

- Clues in the environment may reveal that it's a VM

- Presence of *integration tools*
  - C:\Windows\System32\VBoxTray.exe
  - C:\Program Files\VMware\ VMware Tools\vmtoolsd.exe
  - open-vm-tools (Linux package)



- Running services
  - Hyper-V Time Synchronization Service
  - VMware SVGA Helper Service
  - VirtualBox Guest Additions Service

Hypervisor detection – Hardware

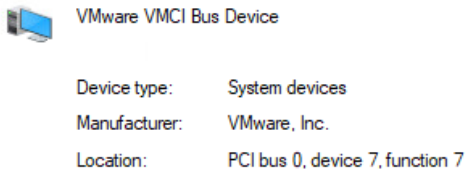
- The hardware usually reveals that it's a VM

- Presence of virtual hardware
  - HDDs
  - Graphics adapters
  - Disk drives

- BIOS/EFI vendor/version strings

```
BIOS Version:                 VMware, Inc. VMW71.00V.16460286.B64.2006250725, 2020. 06. 25.
Windows Directory:            C:\Windows
```

- Special interfaces
  - E.g., VMware VMCI device



Hypervisor detection – Side-channel "attacks"

- Timing- or behaviour-based analysis may also be used to detect a virtualized environment

- Timing
  - Certain CPU instructions may be slower (e.g., CPUINFO)
  - Certain interrupts might take longer

- Behaviour
  - Certain instructions may work differently
  - Caching might work differently
  - "Expected" CPU bugs might not be present (CPU Errata)

Denial of Service attacks

- An attacker may attempt to overload or disrupt the host to cause performance degradation or service outage affecting multiple virtual machines
  - But it may also be an unintended side effect of a legitimate user

- CPU starvation
  - Aims to exhaust all the computing power on the host
    - » Heavy instructions (e.g., AVX, AVX2)
    - » Instructions that cause lots of context switches and interrupts

- Memory starvation
  - If memory is overprovisioned on the VMs, the attacker may be able to cause thrashing on the VMs and the host

Denial of Service attacks

- Disk performance degradation
  - HDDs can become very slow if given random reads/writes
  - Flash-based devices may get physically damaged
- Data loss/corruption: wasting all the space (if overprovisioning)
- Network disruption
  - The attacker may cause network outages by spoofing various packets (ARP, STP, …) → network security
- Countermeasures
  - Setting proper weights limits
  - Careful overprovisioning
  - Auditing and alerting

Unauthorized access

- Individuals may access data without permission on a VM host
  - Disgruntled employees
  - Hackers with stolen (admin) credentials
  - Through vulnerabilities in the virtualization software
- DoS: turning off or deleting VMs and virtual hard disks [ransomware]
- Data theft
  - Dumping a VM's memory to look for valuable information
    - » Private keys, certificates, passwords in flight, …
  - Reading data on virtual hard disks
  - Sniffing (potentially unencrypted) network traffic on the internal network

Unauthorized access

- Data manipulation
  - Altering VM memory
  - Modifying virtual hard disks
  - Spoofing network traffic, lateral movement
- Cloud environments are even riskier
- Countermeasures
  - Proper physical and virtual access control
  - Auditing
  - Regular backups
  - vTPMs (virtual TPMs)
  - Shielded VMs (Microsoft Hyper-V), Encrypted VMs (VMware ESXi)

Virtual Machine escape

- Attacks that aim to escape from the virtualized (and controlled) environment to gain access to
  - Other VMs
  - The host

- Via
  - Exploiting bugs in the VMM or its components
  - Exploiting bugs in the underlying hardware

Virtual Machine escape

CVE-2017-0109 | Windows Hyper-V Remote Code Execution Vulnerability

Security Vulnerability

Published: 03/14/2017
MITRE CVE-2017-0109

A remote code execution vulnerability exists when Windows Hyper-V on a host server fails to properly validate input from an authenticated user on a guest operating system. To exploit the vulnerability, an attacker could run a specially crafted application on a guest operating system that could cause the Hyper-V host operating system to execute arbitrary code.

An attacker who successfully exploited the vulnerability could execute arbitrary code on the host operating system.

3a. Use-after-free vulnerability in SVGA device (CVE-2020-3962)

Description
VMware ESXi, Workstation and Fusion contain a Use-after-free vulnerability in the SVGA device. VMware has evaluated the severity of this issue to be in the Critical severity range with a maximum CVSSv3 base score of 9.3.

Known Attack Vectors
A malicious actor with local access to a virtual machine with 3D graphics enabled may be able to exploit this vulnerability to execute code on the hypervisor from a virtual machine.

Virtual Machine escape

CVE-2018-2698

Published: **18 January 2018**

Vulnerability in the Oracle VM VirtualBox component of Oracle Virtualization (subcomponent: Core). Supported versions that are affected are Prior to 5.1.32 and Prior to 5.2.6. Easily exploitable vulnerability allows low privileged attacker with logon to the infrastructure where Oracle VM VirtualBox executes to compromise Oracle VM VirtualBox. While the vulnerability is in Oracle VM VirtualBox, attacks may significantly impact additional products. Successful attacks of this vulnerability can result in takeover of Oracle VM VirtualBox. CVSS 3.0 Base Score 8.8 (Confidentiality, Integrity and Availability impacts). CVSS Vector: (CVSS:3.0/AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H).

10 new VM escape vulnerabilities discovered in VirtualBox

by **James Sanders** in **Security** on January 25, 2018, 7:25 AM PST

While virtualization platforms are intended to provide full isolation between guest and host operating systems, VM escape vulnerabilities have seen increased scrutiny

Understanding the Scope of Venom (CVE-2015-3456)

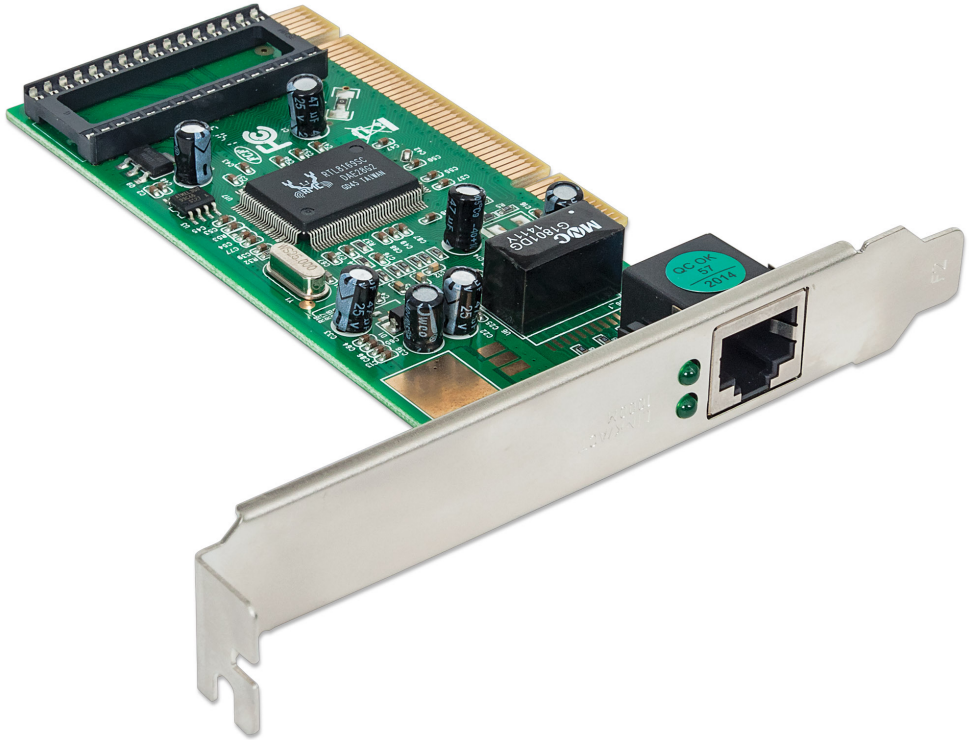
Vulnerability

The Venom vulnerability refers to a flaw in the QEMU FDC (floppy disk controller), which is enabled in the default configuration of **Xen and KVM** virtualization platforms. QEMU is an open-source package with widespread adoption, the scope of which is not fully quantifiable. This vulnerability can be triggered even in configurations in which a floppy device is not present or configured. The flaw potentially allows an attacker to "escape" a guest virtual machine host and gain access to and control of the physical host. This is a privilege escalation vulnerability. In a virtualized environment where the guest can escape to the host, this threat takes on a greater impact. In a successful exploitation, the attack could potentially execute arbitrary code on the underlying host, read and monitor memory contents (exposing sensitive data from guest VMs), or invoke a denial of service on the host (again, affecting all guests).

Hyperjacking

- An attack that aims at
  - (Silently) take over an existing hypervisor
  - Install a second hypervisor below (or above) the legitimate one

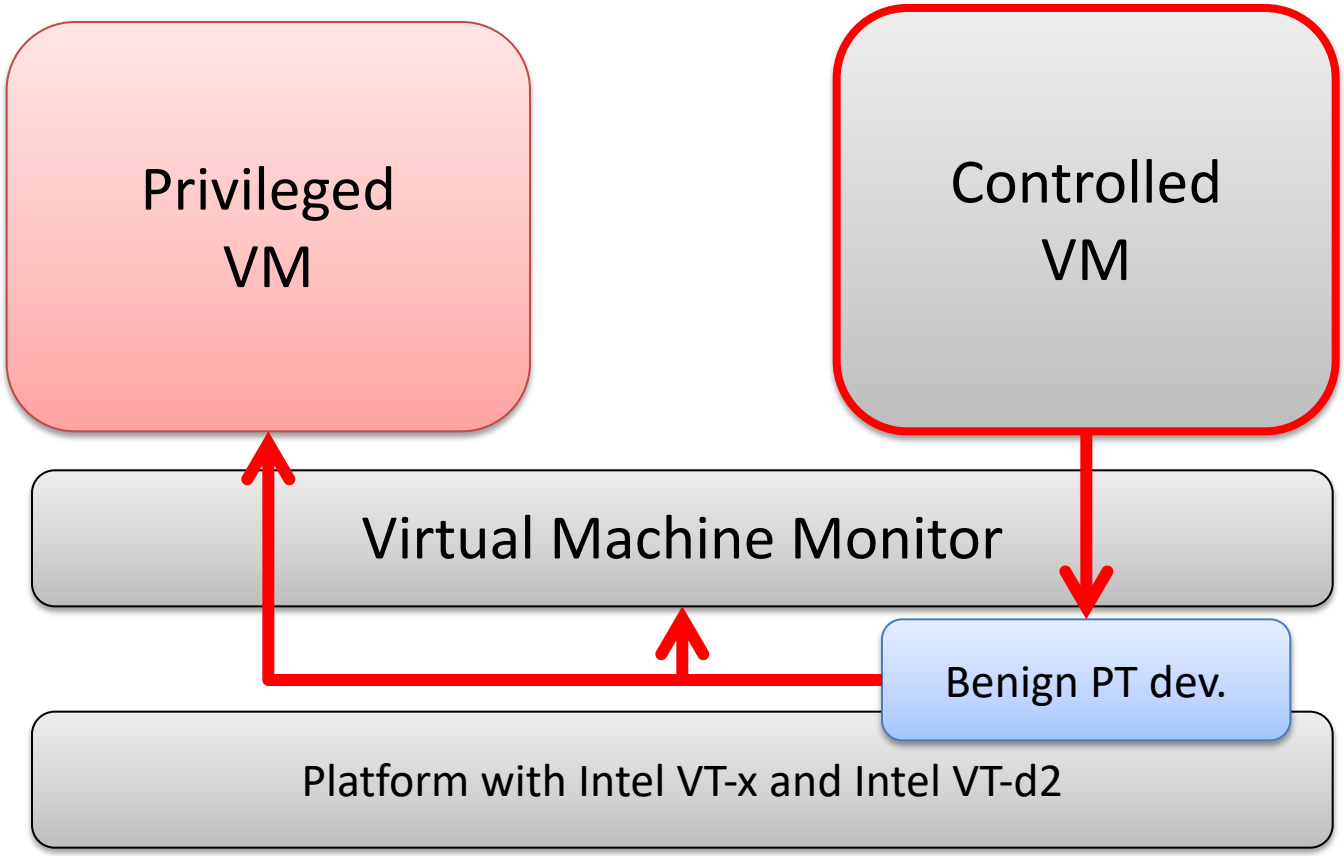
- Similar to rootkits, but even more difficult to detect
- May follow a virtual machine escape



Source: portal.icintracom.com

TASTING I/O VIRTUALIZATION ATTACKS

Threat Model of I/O Virtualizaion



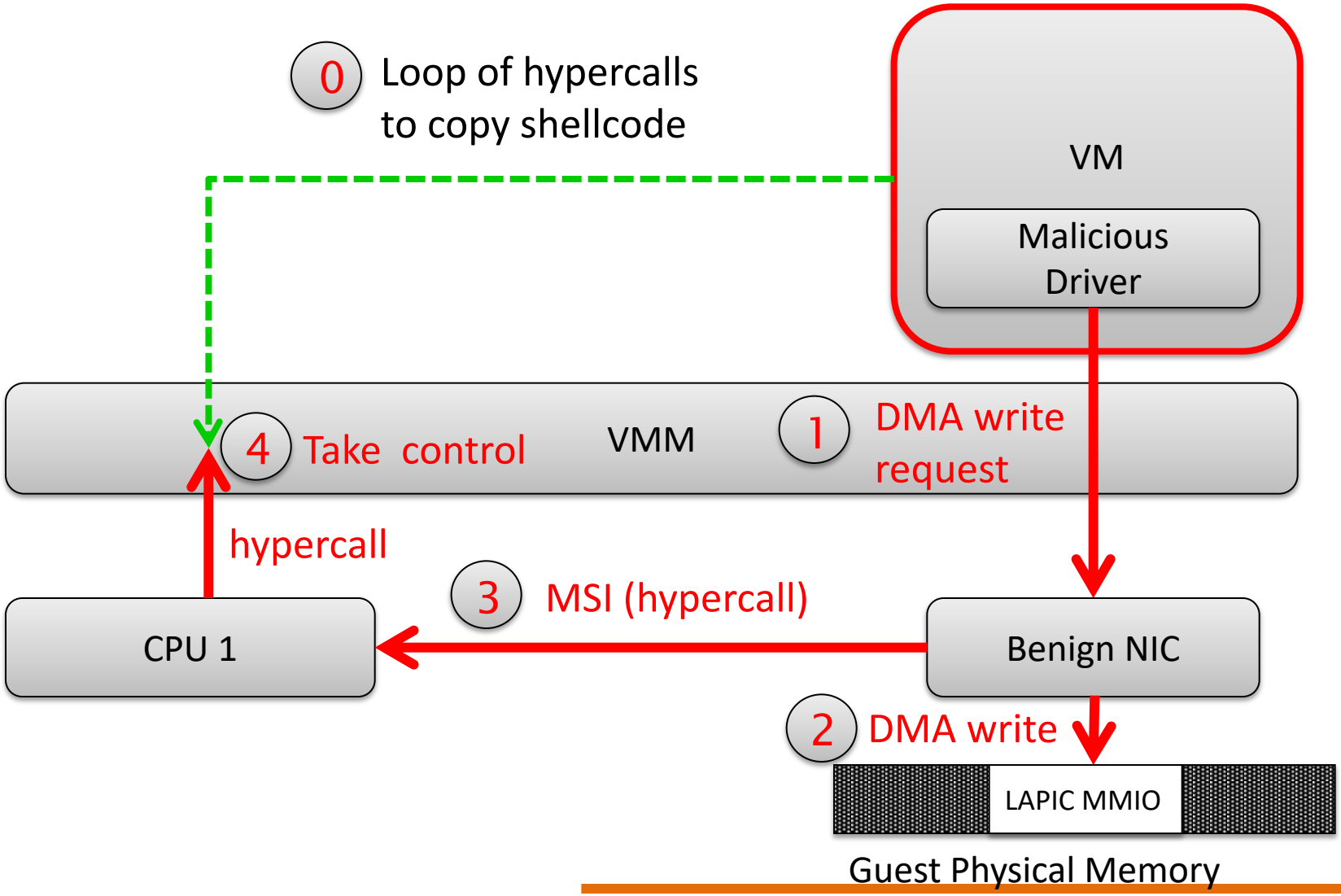
Interrupt remapping

- Recent PCIe devices generate interrupts by using the Message Signalled Interrups (MSI).

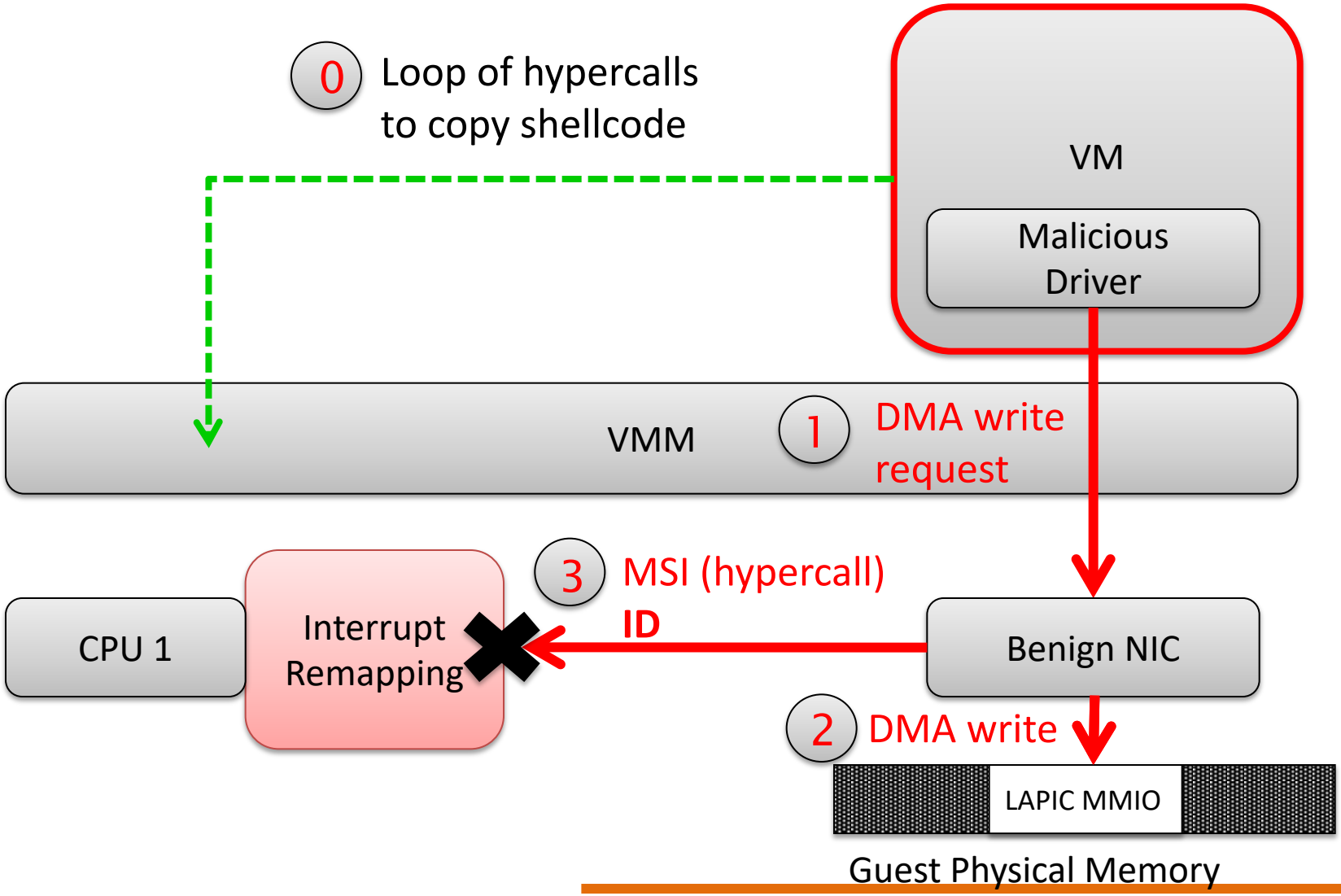
- MSIs can also be generated by enforcing a device's scatter-gather mechanism during a direct memory access (DMA) transaction
  - by writing to a specific MMIO space that belongs to the LAPIC (0xfeexxxxx).
  - LAPIC: Local Advanced Programmable Interrupt Controller
  - MMIO: Memory-mapped I/O

- Theoretical and ***practical*** attacks via Message Signalled Interrupts (MSI)

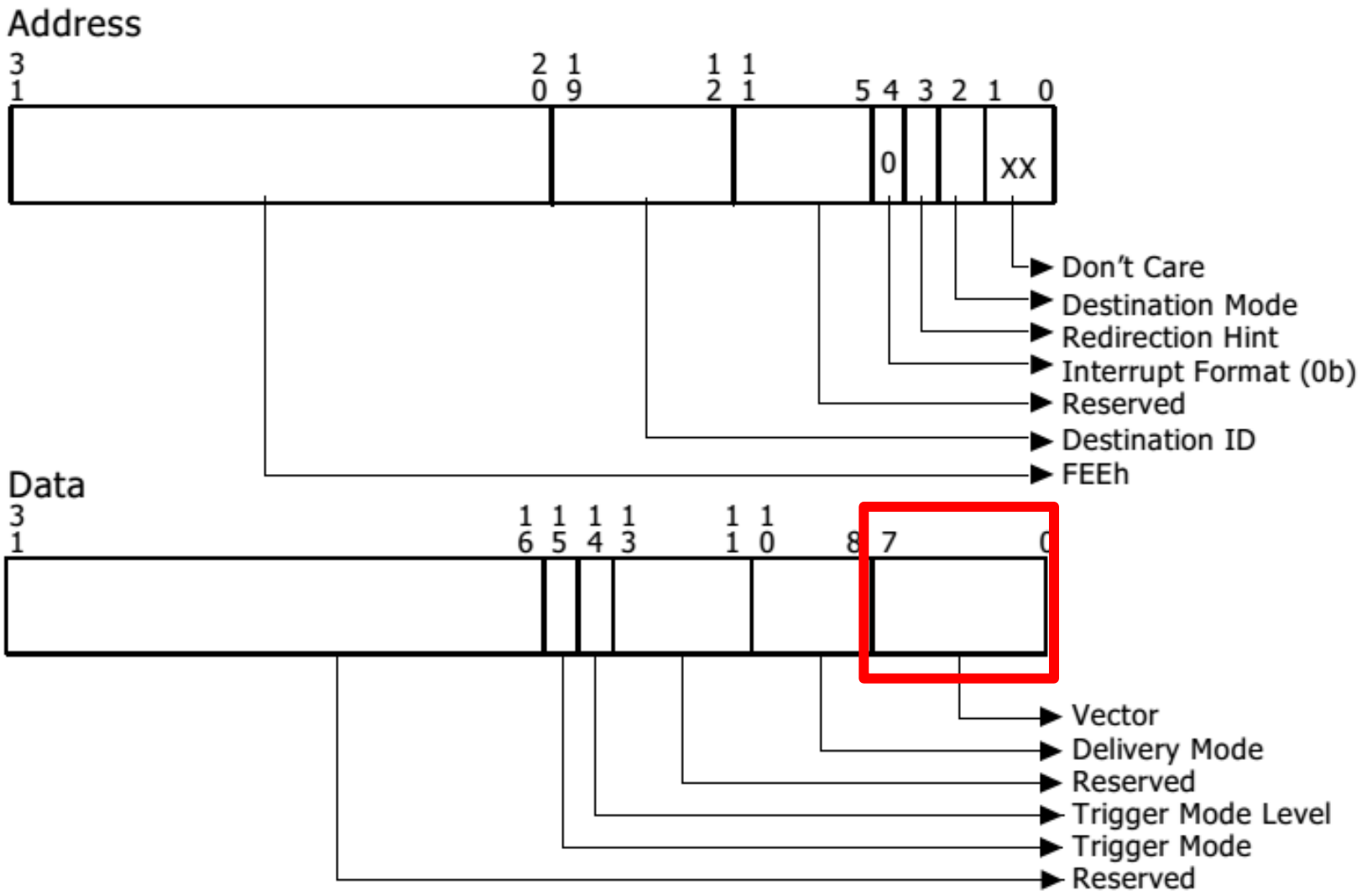
Example: Syscall Interrupt Attack (simplified)



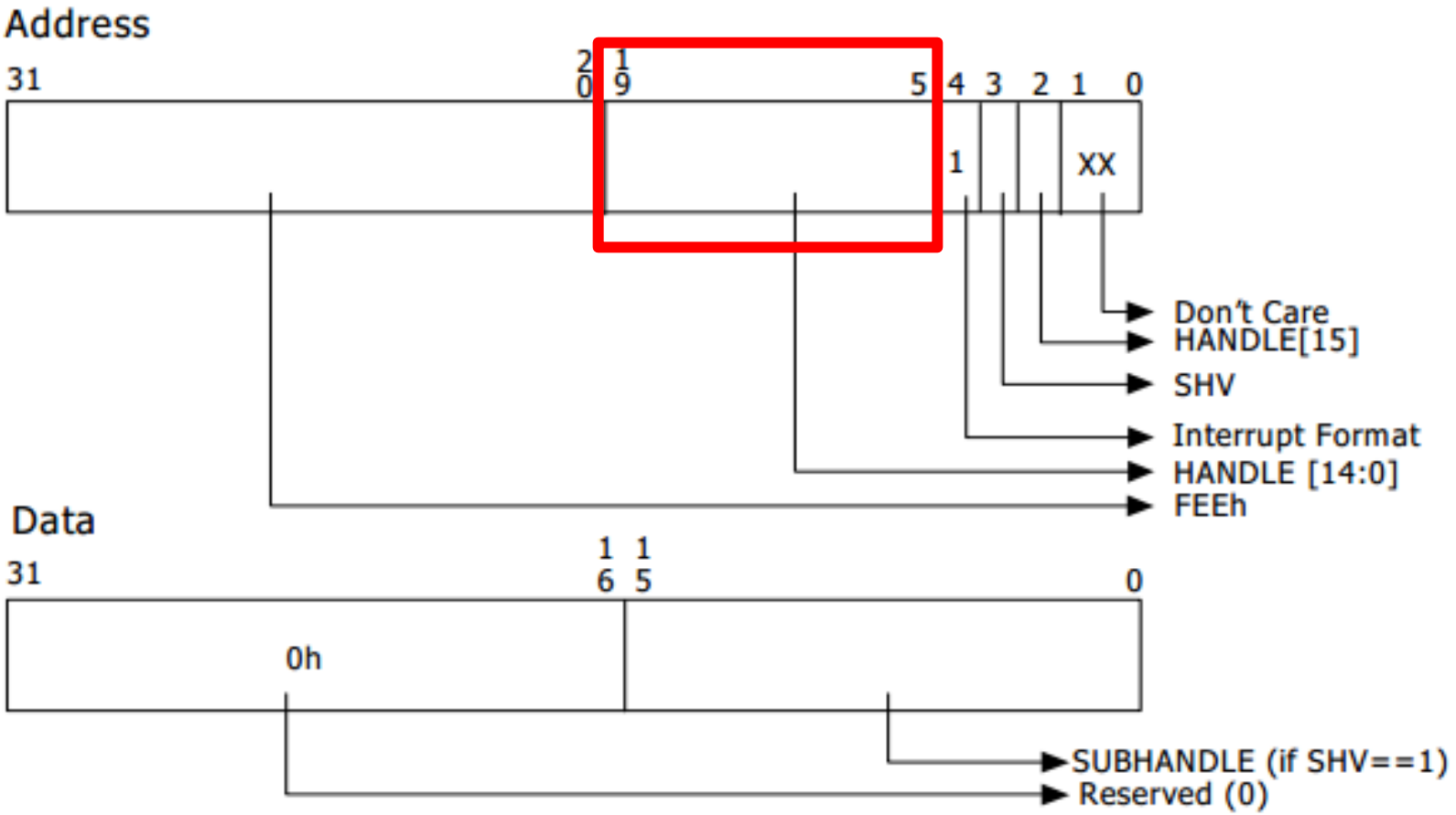
Example: Syscall Interrupt Attack stops



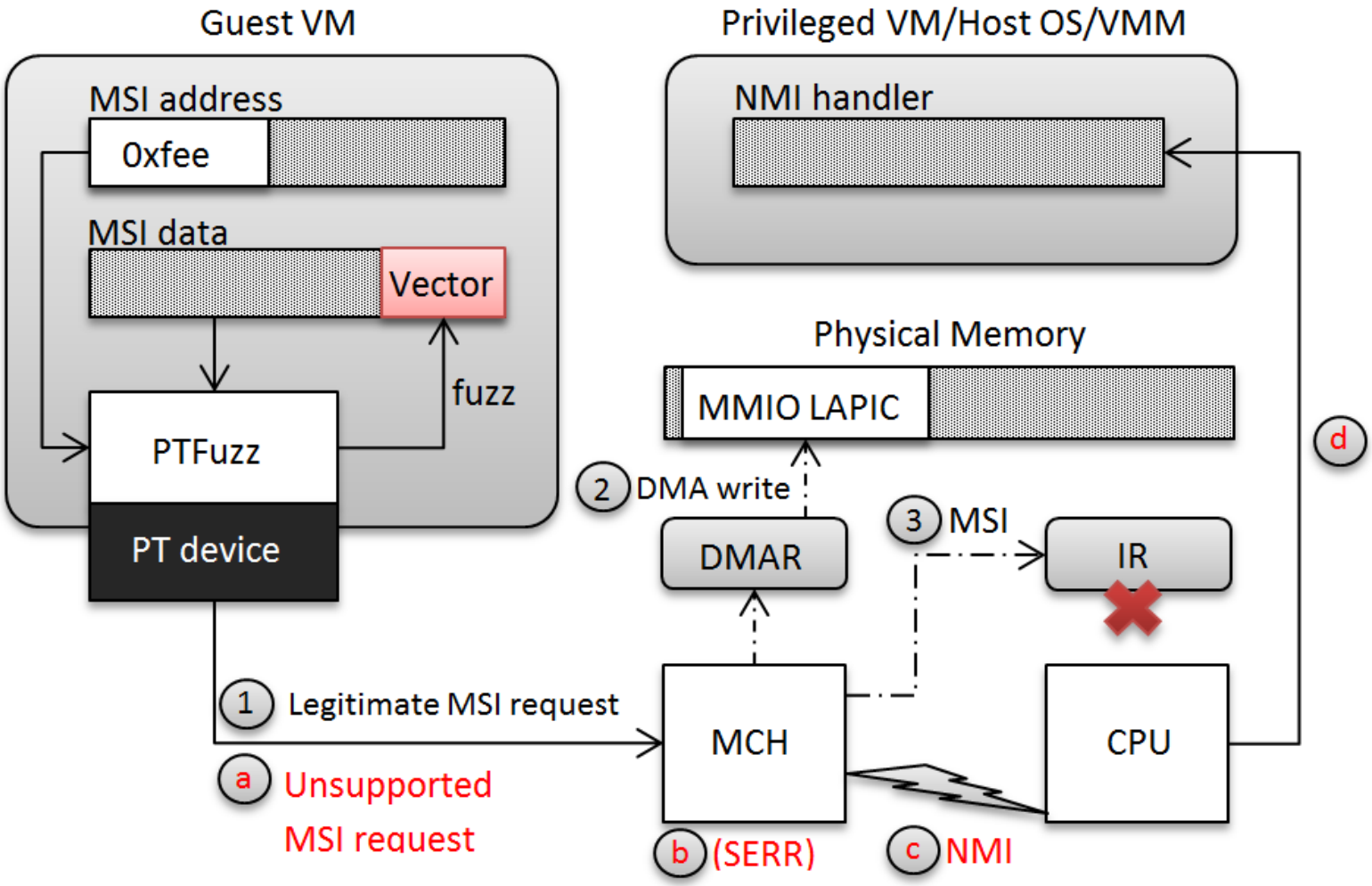
Compatibility-format MSIs



Remappable-format MSI



Generating SERR NMI by fuzzing (CVE-2013-3495)



Indirect DoS

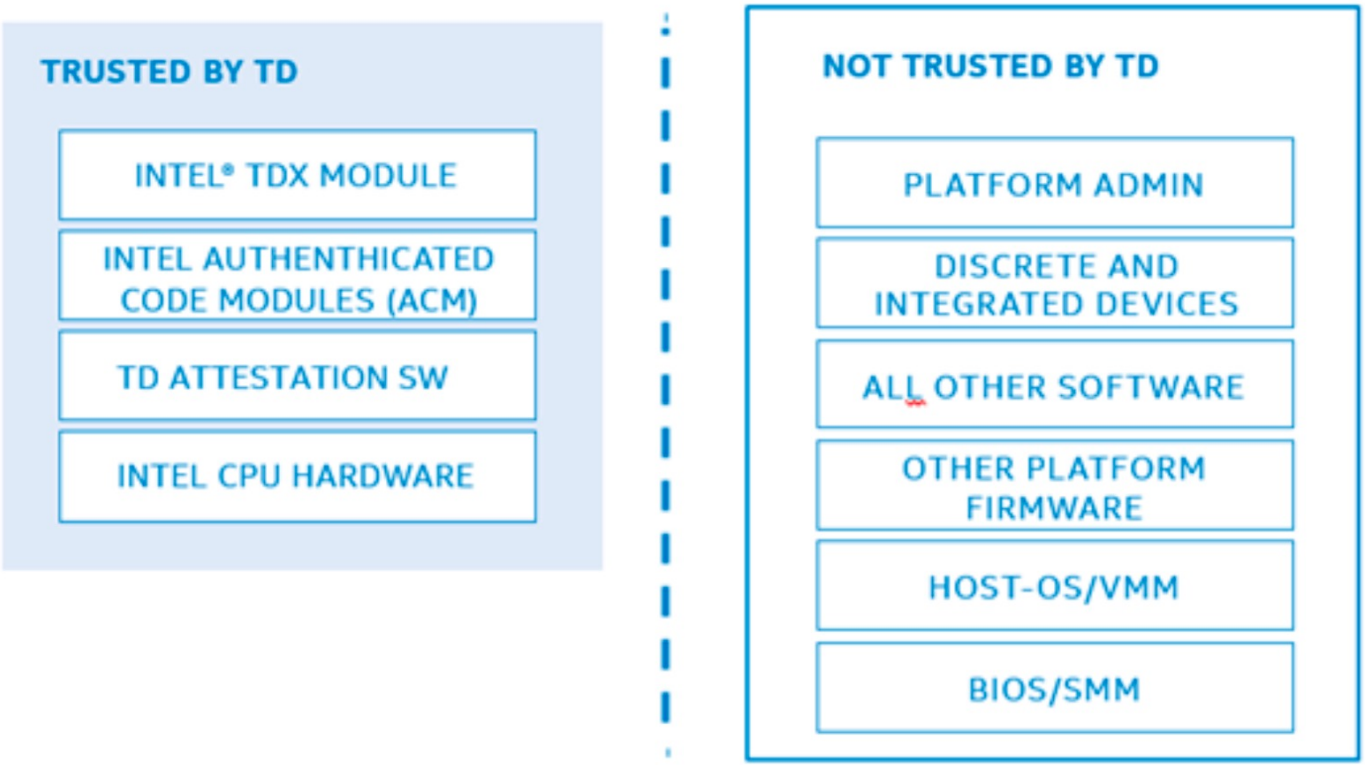
A ***system (hardware) error*** is generated by ***software***

No sysadmin wants such buggy ”hardware” behaviour

PROTECTING HARDWARE VIRTUALIZATION

Intel TDX

- Intel Trust Domain Extensions (Intel TDX) is an architectural technology to deploy hardware-isolated, Virtual Machines (VMs) called Trust Domains (TDs).



Source: https://www.intel.com/

Intel TDX Technical overview

- Designed to isolate TD VMs from the Virtual Machine Manager (VMM), hypervisor and other non-TD software on the host platform.

- Augments defence of the Trust Domains against limited forms of attacks that use
  - physical access to the platform memory, such as offline, dynamic-random- access memory (DRAM) analysis (*e.g.*, cold-boot attacks) and active attacks of DRAM interfaces, including capturing, modifying, relocating, splicing, and aliasing memory contents.
  - Intel TDX does not defend against replay of memory through physical attacks.


Source: https://www.opensourceforu.com/wp-content/uploads/2017/09/Containerisation.jpg

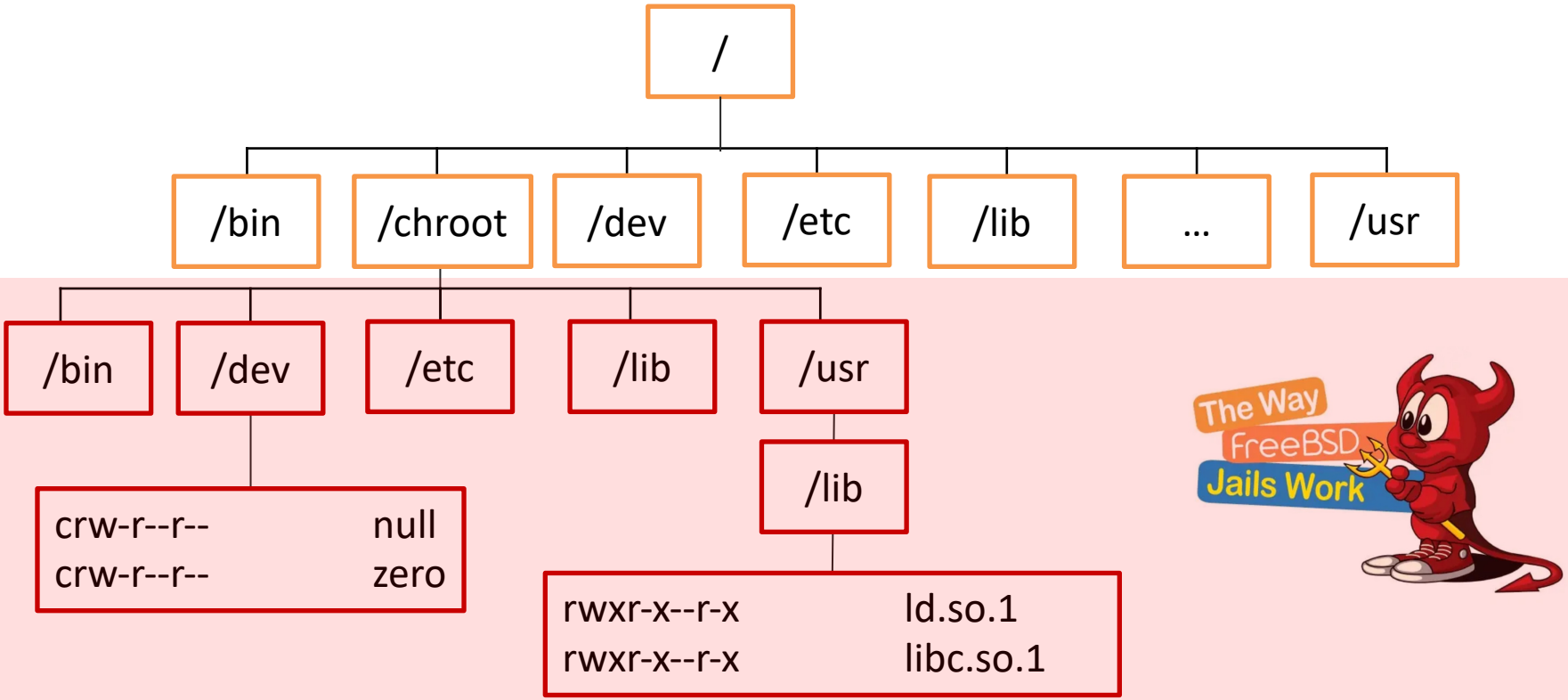
MILESTONES OF CONTAINARIZATION HISTORY

Chroot Jail

- Chroot Jail and the Chroot system call were introduced during the development of Version 7 Unix in 1979
  - Chroot jail is for “Change Root” and it’s considered as one of the first containerization technologies.
  - It allows you to isolate a process and its children from the rest of the operating system.
  - The only problem with this isolation is that a root process can easily exit the chroot.
  - It was never intended as a security mechanism.

FreeBSD Jail

- The FreeBSD Jail was introduced in FreeBSD OS in the year 2000
  - It was intended to bring more security to the simple Chroot file isolation.
  - Unlike the Chroot, FreeBSD implementation isolates also the processes and their activities to a particular view of the filesystem.



Cgroups (Control groups)

- Control Groups provide a mechanism for
  - aggregating/partitioning sets of tasks,
  - and all their future children, into hierarchical groups with specialized behaviour.

- Allows one to allocate resources among user defined groups of processes. Resources include:
  - CPU time
  - System memory
  - Disk I/O
  - Network bandwith, etc.

- CGroups was mainlined into the *Linux kernel in 2007* and is a fundamental part of modern containerization technologies like Docker, Kubernetes, and LXC.

More about cgroups

- They implement resource accounting and limiting.
  - They provide many useful metrics, but they also help ensure that each container gets its fair share of memory, CPU, disk I/O.
  - More importantly, that a single container cannot bring the system down by exhausting one of those resources.

- They do not play a role in preventing one container from accessing or affecting the data and processes of another container,
  - Mostly against DoS attacks.

Linux kernel namespaces

- Namespaces provide the first and most straightforward form of isolation:
  - processes running within a container cannot see, and even less affect, processes running in another container, or in the host system.
  - namespaces were introduced between kernel version 2.6.15 and 2.6.26. (July 2008)

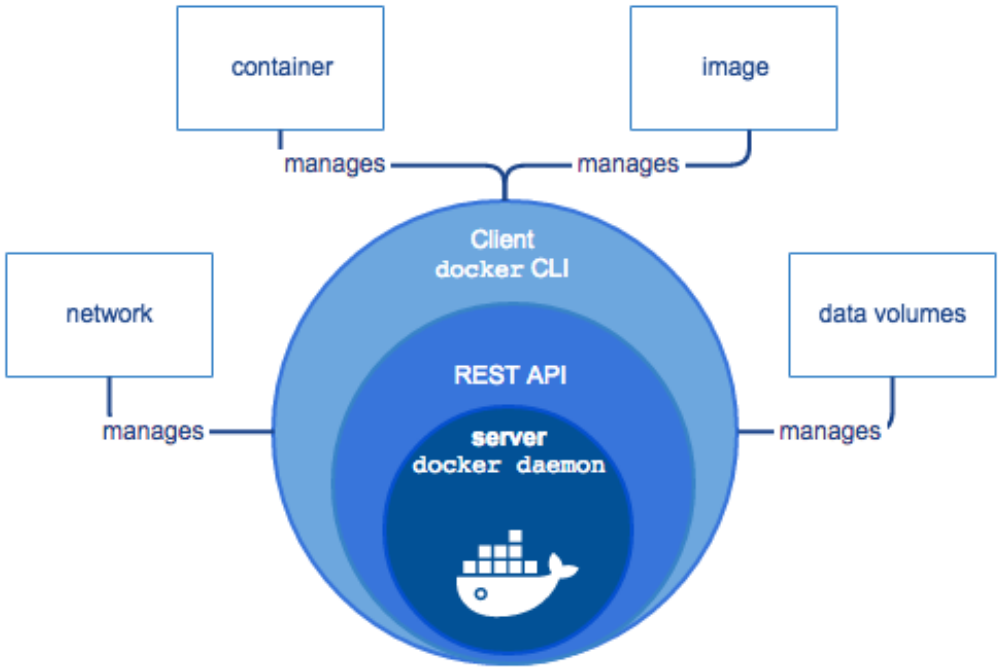
- Each container also gets its own network stack
  - a container doesn't get privileged access to the sockets or interfaces of another container.
  - if the host system is setup accordingly, containers can interact with each other through their respective network interfaces (just like they can interact with external hosts).

LXC

- LXC is a userspace interface for the Linux kernel containment features.
  - Creates an environment as close as possible to a standard Linux installation without the need for a separate kernel.
- Umbrella project atop
  - Kernel namespaces (ipc, uts, mount, pid, network and user)
  - Apparmor and SELinux profiles
  - Seccomp policies
  - Chroots (using pivot_root)
  - Kernel capabilities
  - CGroups (control groups)
- Through a powerful API and simple tools, it lets Linux users easily create and manage system or application containers.

Docker

- When Docker started it used LXC as a container runtime
  - the idea was to create an API to manage the container runtime,
  - isolate single processes running applications
  - supervise the container life cycle and the resources it uses.
- Later, Docker started building a monolithic application with multiple features from launching Cloud servers to building and running images/containers.
  - Docker used “libcontainer” to interface with Linux kernel facilities like Control Groups and Namespaces.



DOCKER ENGINE SECURITY

Aspects of Docker security

- The intrinsic security of the kernel and its support for namespaces and cgroups
  - "hardening" security features of the kernel and how they interact with containers
- The attack surface of the Docker daemon itself
- Loopholes in the container configuration profile
  - either by default, or when customized by users
  - content trust issues

Linux kernel: capabilities

- Capabilities turn the binary “root/non-root” dichotomy into a fine-grained access control system.
- Processes (like web servers) that just need to bind on a port below 1024 do not need to run as root:
  - they can just be granted the net_bind_service capability instead.
- And there are many other capabilities, for almost all the specific areas where root privileges are usually needed.
- By default, Docker drops all capabilities except those needed, an allowlist instead of a denylist approach.

Additional kernel security features

- Capabilities are just one of the many security features provided by modern Linux kernels.

- It is also possible to leverage existing, well-known security solutions like AppArmor, SELinux, GRSEC, etc. with Docker
  - Docker currently only enables capabilities; it doesn't interfere with the other systems.

- You can run a kernel with GRSEC and PAX.
  - This adds many safety checks, both at compile-time and run-time.
  - It also defeats many exploits, thanks to techniques like address randomization.
  - It doesn't require Docker-specific configuration, since those security features apply system-wide, independent of containers.

Docker daemon attack surface

- The Docker daemon requires root privileges.
  - Unless you opt-in to Rootless mode (available since v20.10) and install the prerequisites
- Only trusted users should be allowed to control your Docker daemon.
  - Docker allows you to share a directory between the Docker host and a guest container.
  - It allows you to do so without limiting the access rights of the container.
- If you instrument Docker from a web server to provision containers through an API, you should be even more careful than usual with parameter checking.
  - You can also expose the REST API over HTTP if you explicitly decide to do
    - » it is mandatory to secure API endpoints with HTTPS and certificates.

Docker Vulnerability Scanning with Snyk

- Vulnerability Scanning allows developers and development teams
  - to review the security state of the container images
  - take actions to fix issues identified during the scan,
  - resulting in more secure deployments.

snyk

- The scan result includes
  - the source of the vulnerability, such as OS packages and libraries, version in which it was introduced (CVE),
  - and a recommended fixed version (if available) to remediate the vulnerabilities discovered.
- Vulnerability scanning for Docker local images can run on Snyk engine, providing users with visibility into the security posture of their local Dockerfiles and local images.
  - When you push an image to Docker Hub after enabling vulnerability scanning, Docker Hub automatically scans the image to identify vulnerabilities in your container images.

SERVERLESS SECURITY

Short overview of Serverless

- Perform quite simple processing of events without maintaining underlying infrastructure.
  - Service of cloud providers (e.g., AWS, Google, etc)
- An event-driven model
  - applications aren't always running but instead are triggered by events.
  - Ideal for short-term jobs (pay per use)

AWS Lambda

- Lambda function
  - AWS packages it into a container and executes the container on a multitenant cluster of machines

OWASP Serverless top 10

- A1:2017 Injection
- A2:2017 Broken Authentication
- A3:2017 Sensitive Data Exposure
- A4:2017 XML External Entities (XXE)
- A5:2017 Broken Access Control
- A6:2017 Security Misconfiguration
- A7:2017 Cross-Site Scripting (XSS)
- A8:2017 Insecure Deserialization
- A9:2017 Using Components with Known Vulnerabilities
- A10:2017 Insufficient Logging and Monitoring

Example: A1:2017 Injection

- Same story: SQL/noSQL injection, Code injection
- Serverless functions can also be triggered from different event sources (and not only from the API) such as:
  - cloud storage events (e.g., S3, Blob and other cloud storage),
  - stream data processing (e.g., AWS Kinesis),
  - databases changes (e.g., DynamoDB, CosmosDB),
  - code modifications (e.g., AWS CodeCommit) notifications (e.g., SMS, Emails, IoT)
- No longer control of the line between the origin to the resource.
  - If a function is triggered via email or a database, where to put controls such as the Firewall?

Further reading – Virtual Machines

- [1] Pék, G., Buttyán, L. and Bencsáth, B., 2013. *A survey of security issues in hardware virtualization*. ACM Computing Surveys (CSUR), 45(3), pp.1-34.
- [2] Rafal Wojtczuk and Joanna Rutkowska. *Following the White Rabbit: Software attacks against Intel VT-d technology*, April 2011.
- [3] Pék, G., Lanzi, A., Srivastava, A., Balzarotti, D., Francillon, A. and Neumann, C., 2014, June. *On the feasibility of software attacks on commodity virtual machine monitors via direct device assignment*. In Proceedings of the 9th ACM symposium on Information, computer and communications security (pp. 305-316).
- [4] Intel® Trust Domain Extensions (Intel® TDX) https://cdrdv2.intel.com/v1/dl/getContent/690419

Further reading - Containers

- [5] https://medium.com/faun/the-missing-introduction-to-containerization-de1fbb73efc5

- [6] https://docs.docker.com/engine/security/
- [7] https://docs.docker.com/engine/security/trust/

- [8] https://man7.org/linux/man-pages/man7/capabilities.7.html

- [9] https://docs.docker.com/engine/scan/
- [10] https://docs.docker.com/docker-hub/vulnerability-scanning/

Further reading - Serverless

- [11] https://cloudinfrastructureservices.co.uk/aws-lambda-vs-containers-whats-the-difference/
- [12] https://aws.amazon.com/lambda/
- [13] https://cloud.google.com/functions/
- [14] https://owasp.org/www-project-serverless-top-10/

THANK YOU. QUESTIONS?