HYPERVISOR VULNERABILITY RESEARCH

TRAINING OVERVIEW

This professional deep technical training offers a systematical foundation for low-level security research of arbitrary hypervisor systems, taught from an offensive security research perspective. The focus of this "Level I" training is on effective vulnerability discovery and analysis.

The training brings together a wide scope of knowledge exposure with deep dives and practical experience, compressed with aggressive systematization and specifically optimized for accelerated learning. Topics such as fundamental theory, generalized design models, practical threat models, real-life vulnerability case studies and implementation-specific technical details form the conceptual basis of the training materials.

All exercises and labs in this training are based on real life tasks, and target essential vulnerability research skills. Training materials are updated and refreshed prior to each live class.

AUDIENCE

This training is primarily intended for professional security researchers and virtualization systems engineers.

Materials of this training constitute an essential base of knowledge for an effective zero day vulnerability researcher targeting any hypervisor implementation.

Training approach and materials are suitable for complete beginners with application security and hypervisors.

PREREQUISITES

Essential:

- C/C++ (reading level);
- Linux command line and build environment.

Recommended:

- x86_64 assembly language;
- some experience with vulnerability research;
- basic OS and hardware design concepts.

PLAN

Day 1. Foundation.
Virtualization technology fundamentals. Virtual machine guest services. Oracle VirtualBox.

Day 2. Virtual hardware.
Virtual devices subsystem. OS & hardware design theory. VMware Workstation.

Day 3. Paravirtualization.
Hardware virtualization technology. Hypercalls and paravirtualized devices. Microsoft Hyper-V.

Day 4. Hypervisor core.
Virtual Machine Monitor (VMM). CPU & MMU theory. Linux KVM.

LEVELS & CERTIFICATION

Complexity level: intermediate-advanced.
Certificate grade: C0DE1 (Zero Day Engineering).
Next level training: "Advanced Hypervisor Exploitation".

TRAINING DETAILS

All modern hypervisor systems share a common foundation of conceptual models: design models, threat models, virtual hardware models – that may be abstracted away from the technical details of a particular implementation. Key principle of this training is to introduce the students to vulnerability research in arbitrary hypervisors through the perspective of their common abstractions, while leveraging implementation-specific technical case studies and the author's many-year experience in operating systems introspection and vulnerability research to facilitate practical applications.

The subject of this training – system internals, vulnerability patterns and trends – represents an essential basis for all offensive and defensive workflows in software security production: such as vulnerability discovery, exploit development, secure systems design, writing safe code and security auditing, as well as ensuring good decisions with 3rd party products selection, deployment and integration in security-critical environments.

The structure of this training is an even mix of theoretical and hands-on experience. Training materials are systematically designed, with each training day focused on one common subsystem of a hypervisor, and divided into four self-contained two-hour blocks that cover relevant theoretical concepts, threat models and attack vectors, known vulnerability patterns and trends, and key technical details of several popular hypervisor implementations.

Exercises in this training are based on suitable real-life tasks and specially crafted with time management techniques in order to enable accelerated learning of skills and internalization of extensive knowledge. Practical environments are based mostly on static source code analysis with occasional binary analysis and debugging labs.

Knowledge presented in this training is based on independent technical research and reverse engineering work done by the author herself, and illustrated with security vulnerabilities discovered by the author, as well as with publicly available sources. Training materials are updated prior to each training session to incorporate important public advancements in the field and new technological developments.

This training is primarily focused on four popular hypervisor implementations: Oracle VirtualBox, VMware Workstation, Microsoft Hyper-V, and Linux KVM. Other virtualization and emulation systems, though not discussed in a dedicated manner, are occasionally featured in specific vulnerability case studies.

ABOUT THE AUTHOR

Alisa Esage is a professional low-level code breaker, vulnerability researcher and reverse engineer. She was credited by Microsoft, Google, Firefox, Oracle, Schneider Electric, and other leading software vendors for discovery of previously unknown security bugs. She is specialized on attacking popular and hardened software and firmware implementations for exploit development.

For several years Alisa focused on modern virtualization security and system internals, working on discovery and exploitation of vulnerabilities in multiple popular hypervisors. In the process she developed a deeply generalized perspective on attacking modern hypervisors, which forms the basis of this training. Alisa is the winner of Pwn2Own 2021 competition in the Virtualisation category.

LEARNING OBJECTIVES

Upon completion of this training class the students are expected to have obtained the following knowledge and skills:

- A deeply systematical and generalized perspective on virtualization systems design and internals;
- Recap of the relevant theoretical concepts from operating systems theory, hardware design and application security;
- Familiarity with key implementation-level peculiarities and general design of several major hypervisor products, both open source and proprietary;
- Knowledge of the classes, patterns and trends of software vulnerabilities which are specific to hypervisors, based on deep technical details of several dozens of real-life case studies, and an overview of the publicly exposed vulnerability history;
- Fundamental practical skills of vulnerability discovery and prototyping in large and popular code bases, based on modern toolsets and the author's personal playbook;
- Ability to discover previously unknown vulnerabilities in arbitrary hypervisor software systems;
- Ability to prototype a known hypervisor vulnerability based on a security patch;
- A deep sense of trends and future developments in hypervisor security research.

HARDWARE & SOFTWARE REQUIREMENTS

All students are required to prepare their own lab and exercise environment for the class.

Hardware and software requirements:

1. Productivity-grade laptop or a desktop computer with a modern HVT-enabled chipset (either Intel VT-x or AMD-V).
2. A modern Linux OS installation, either:
    a. A modern Linux native installation, or
    b. A modern Linux virtual machine with nested virtualization enabled (see Notes).
3. A professional disassembler.
4. An advanced IDE with C/C++ syntax intelligence.

Notes:

- Hands-on labs were tested on Ubuntu Linux 20.04 LTS x64. In case of a different Linux platform flavor, the student would be responsible to specialize the build instructions and introspection guides to it.
- Linux environment will be utilized (among other things) to build VirtualBox and run a virtual machine in the self-build. Unless Linux is running on bare metal, that would require nested virtualization enablement in the base hypervisor.
- Majority of modern desktop hypervisors support HVT pass-through ("nested virtualization") on both Intel and AMD chipsets, which is required for option 2b. Software recommendations for the latter option: VirtualBox, VMware Workstation, Parallels Desktop.

TRAINING PACKAGES & FEES

This training is available in several formats: live and self-paced, public and private, and various package options.

Online training is based on a modern streaming platform with high-quality audio and video, and a group chat. The instructor will be available for questions, feedback and technical support only for students with Advanced package.

Self-paced training offers exactly the same learning experience as the online training, less the mentoring aspect and instant interactivity.

All training packages were specifically optimized for online experience.

LIVE ONLINE TRAINING

Basic package

What's included:
- Access to the public online training.
- Training slides and materials.
- Training completion certificate.

Price: €3,900.- per seat.

Advanced package

What's included:
- Everything in the Basic package.
- Personal feedback and technical support from the instructor during the training.
- A possibility to receive a Training Achievement certificate.

Price: €4,200.- per seat.
Limited seats.

SELF-PACED TRAINING

Basic package

What's included:
- Video lectures, exercises, and walk-through.
- Training slides and materials.
- Training completion certificate.

Price: €2,900.- per seat.

Complete package

What's included:
- Everything in the Basic package.
- One month of technical support by email.
- An on demand personal consultation with the instructor by video call.
- Join our online public training on the same topic during the year at no additional cost.

Price: €4,200.- per seat.

PRIVATE & CUSTOM TRAINING

Minimum group size for a private training is 10 persons. Contact us to check availability.

LIVE TRAINING DATES & BOOKING

Refer to our website for the dates of the nearest public class. This training is primarily held on, and optimized for, ONLINE streaming platforms.

All our online training courses are offered exclusively at the Zero Day Engineering project (zerodayengineering.com).

Contact us directly for all the bookings and purchases.

TRAINING PROGRAM

DAY 1. FOUNDATION

1.1. Virtualization 101.

Virtualization technology flavors. Emulation, binary translation, recompilation, raw execution, sandboxing, hardware assisted virtualization. Bare-metal and hosted hypervisors. Hypervisor subsystems: VMM, virtual devices, guest services, hypercall interfaces. Modern trends in development and implementation of hypervisors.

Agenda:
- History of virtualization technology.
- Virtualization technology landscape.
- Terminology and technologies.
- General design model of a hypervisor.
- Threat models, attack surfaces and attack vectors.

Exercise: investigate a popular implementation of a virtualization system in source code.
Target skill: initial orientation in a large source code base.

1.2. Guest services subsystem.

Rich virtualization functionality: shared folders, clipboards, smart cards. Drag-and-drop, virtual printer, memory ballooning. Hardware accelerated 3d & 2d graphics and graphic shaders, GPU virtualization, OpenGL and DirectX pass-through. Networking services: DHCP, TFTP, zero-conf, PXE boot. NAT, virtual LAN, inter-VM networking.

Agenda:
- Overview and functions of guest services.
- Relevant OS theory.
- Implementation options.
- Threat models and attack vectors.
- Common security issues.

Exercise: investigate an implementation of guest services in source code.
Target skill: identifying and navigating a specific subsystem in a large code base.

1.3. Attacking guest services.

Agenda:
- Threat models and attack vectors.
- Principles of fuzzing and static analysis.
- Examples of known vulnerabilities in guest services.

Exercise: security patch analysis based on source code.
Target skill: identifying and understanding a security vulnerability by analyzing source code modifications.

1.4. VirtualBox implementation.

Agenda:
- Overview of design and implementation.
- Guest additions / virtualization tools.
- Building instructions.
- Security research trends.
- Case study: 3d graphics subsystem.

Exercise: find as many bugs as you can in the given attack vector by static analysis.
Target skill: basic vulnerability discovery by pattern-based static analysis in source code.

DAY 2. VIRTUAL HARDWARE

2.1. OS & hardware theory.

CPU privilege modes, the kernel and loadable kernel modules, kernel-userland interfaces. Common classes of computer hardware: processing units, SoCs, buses, peripherals. Fundamental hardware concepts: registers, BARs, I/O, interrupt and transfer management. PIO, MMIO, DMA. FIFO buffers (queues), FILO buffers (stacks), ring buffers, transfer descriptors, interrupts and polling. Device driver functions, skeleton and top-level algorithm.

Agenda:
- OS and hardware ABC.
- Hardware-software boundary and interfaces.
- PCI bus specification.
- USB bus overview.
- Intel 8254xxx (E1000) ethernet controller model.

Lab: live hypervisor introspection.
- Inspecting virtual hardware from within a guest OS.
- Debugging a hypervisor.
- Testing attack vectors.

2.2. Anatomy of a Linux device driver.

Common logical blocks of a Linux kernel module. Device driver specifics. *OPS and OS callbacks. Probe function, device initialization and set up, operation routines. Common callback structures: struct pci_driver, usb_driver, drm_driver. Common hardware resource management primitives: ioremap, pci_*, dma_*, and their Linux subsystems.

Agenda:
- Linux kernel module skeleton and building blocks.
- Kernel management hooks.
- Kernel-userland interfaces.
- Resource management primitives.
- Probe function.

Exercise: creating a vulnerability proof-of concept from a binary security patch, part 1: analysis of a virtual device driver in source code.

2.3. Attacking virtual devices.

Types of virtual devices: purely emulated devices, paravirtualized devices, pass-through, combined models. Common issues: bugs in PIO/MMIO/DMA handling, bugs in parsing of internal command protocols, unsanitized values in transmit descriptors. Integer and buffer overflows, race conditions, TOCTTOU.

Agenda:
- Threat models and attack vectors.
- Research landscape and trends.
- Vulnerability case studies.

Exercise: creating a vulnerability proof-of concept from a binary security patch, part 2: binary vulnerability patch analysis.
Target skill: effective binary patch analysis of an unfamiliar component in a proprietary code base.

2.4. VMware Workstation.

The vmx process, the Backdoor interface, emulated device models. The vmx configuration file, undocumented configuration parameters, useful settings. VMware-specific SVGA and VMCI virtual devices.

Agenda:
- Design and implementation overview.
- Guest additions and virtualization tools.
- Research platform set up instructions and tips.
- Advanced configuration.
- Security research trends.
- Case study: VMware SVGA device.

Exercise: creating a vulnerability proof-of-concept from a binary security patch, part 3: analysis of the virtual device implementation, mapping the vulnerability to software, and contemplating the testcase algorithm.
Target skill: generalizing from a low-level vulnerability view to virtual hardware model and virtual device driver, and putting it all together to create a proof-of-concept.

DAY 3. PARAVIRTUALIZATION

3.1. Hardware virtualization technology.

Overview of Intel VT-x, EPT, and AMD-V virtualization extensions implementations. VMX root operation, VMX non-root operation, VM exit, VM entry, VMCS.

Agenda:
- Intel VT-x: basic concepts.
- Virtual Machine Control Structure (VMCS).
- Programming a hypervisor.
- The hypervisor execution loop.

Exercise: analyze a hardware-assisted hypervisor implementation in source code.
Skill: identifying ultra-specialized low-level functionality in a large code base.

3.2. Hypercalls & paravirtualization.

Hypercall interface implementations: legacy – CPU instruction interception, synthetic emulated device; modern - HVT-based. Paravirtualization vs. emulation. Shared memory and synthetic interrupts.

Agenda:
- A legacy hypercall implementation case study.
- A modern hypercall implementation case study.
- Paravirtualization vs. emulation.
- The shared memory paravirtualization interface.
- Para-protocols.

Exercise: investigate a modern HVT-based hypercall implementation in source code of a hypervisor driver and theoretical specifications.

3.3. Attacking hypercalls & paravirtualization.

Agenda:
- Threat models and attack vectors.
- Common issues.
- Fuzzing hypercalls.
- Vulnerability case studies.

Exercise: theoretical analysis of a relevant real-life vulnerability and mapping it to hypervisor drivers.
Target skill: deduce a particular vulnerability technical details, practical context and research implications from a brief public mention.

3.4. Microsoft Hyper-V implementation.

The hypervisor (hvix64.exe/hvax64.exe), the hypercall interface, the VMBUS. Root partition, child partitions, enlightened and unmodified guest OS, Generation 1 and Generation 2 virtual machines. Emulated and paravirtualized hardware components. Enhanced session mode and the Remote Desktop Services (RDP) infrastructure. VSP/VSC hypervisor driver model, Linux Integration Services and Microsoft Windows Hyper-V drivers. VBS (Virtualization Based Security), WMI (scripting), the Hypervisor API.

Agenda:
- Design and implementation overview.
- Attack surfaces and attack vectors.
- Attack surface reduction in practice.
- Research platform set up.
- Security research trends and tips.
- Case study: Virtual Network Switch.

Exercise: analyze a paravirtualized device driver for purposes of fuzzing.
Target skill: conceptual modeling of a custom specialized fuzzer implementation.

DAY 4. VMM

4.1. CPU & MMU theory.

CPU ISA basics, interrupts and exceptions. Privileged and unprivileged CPU execution modes. Special instructions with respect to virtualization requirements. Conditional and unconditional VM-exits. x86_64: CPUID, GETSEC, INVD, XSETBV. MSR and CRx registers. Pipelines, prefetching, and optimization. MMU hardware operation overview. Paging modes: 32-bit, PAE, 4-level. PML4E, PDPTE, PDE, PTE.

Agenda:
- CPU design and operation software theory.
- Memory address translation process.
- OS kernel management of CPU & MMU hardware operation.

Exercise: theoretical analysis of a VM exit and CPU instruction emulation in source code.

4.2. Virtual Machine Monitor subsystem.

CPU & MMU virtualization in practice. Full CPU emulation, special instruction emulation, partial pass-through of instruction functionality, CPU instruction interception. HVT, binary translation and interception-based handling of special instructions in practice. Virtual MMU: shadow page tables, nested page tables and hardware-assisted MMU virtualization technology (EPT).

- Legacy (software-based) and modern (hardware-assisted) CPU virtualization techniques.
- Legacy and modern MMU virtualization techniques.
- Common VMM designs in practice.

Exercise: theoretical analysis of an MMU virtualization implementation in source code.

4.3. Attacking the VMM.

Common attack vectors: CPU instruction emulation, VM exit handling, nested VMX emulation. Common classes of issues: design issues due to mishandled hardware specification; memory corruptions in instruction emulation; memory page table updates and shadowing sanitization issues. Speculative execution bugs.

Agenda:
- Threat models and attack vectors.
- Offensive research trends.
- Vulnerability case studies.

Exercise: analysis of a speculative execution bug and exploit.
Skill: rapid grasping of a low-level attack concept; paradigm shift from vulnerability discovery to exploit engineering.

4.4. Linux KVM implementation.

Agenda:
- Design and implementation overview.
- KVM API and notable consumers.
- Security research trends.
- Implementation deep dive.

Exercise: create a proof-of-concept testcase for a VM-escape vulnerability based on a security patch in source code, and crash VirtualBox.

Note**:** the training agenda may experience minor changes that won't impact expected outcomes of the training.

WHAT TO EXPECT?

Quoted below are anonymized extracts of private feedback from our students with their subjective impressions that may help prospective attendees to better understand the training experience.

"It was empowering. Not only did I feel like I learned an enormous amount, but by the end I felt confident I knew **how to start looking for real vulnerabilities in virtualization systems**."

"I had an amazing time in the training. I feel like a lot of the **knowledge I had was clarified in the training and is now more organized**. Of course I also **learned a lot of new stuff** and it was really interesting and useful."

"It is a well written training, both the materials/slides and exercises are all **well designed**. I also really appreciated the knowledge you showed in the training, it is clear you have **a lot of experience in hypervisor research** and it was great to learn from you."

"I feel like the fact that a big part of the training was to show **how to research and explain your methodology** was really good, it was useful to learn how to approach a problems/research objective when it comes to different attack vectors."

"I really like the **more technical parts** – e.g. different IO options, how hardware virtualization works, OS ABC, MMU virtualization, I found them more interesting than the **specifics of a certain hypervisor**. Also liked the part where you compare different vulnerability types, and how the type recent vulnerabilities indicate the kind of scrutiny a project has seen."

"[I learned that] finding bugs in virtualization systems is achievable. Before doing this course **hypervisor exploitation seemed like an unknowable thing** that was just "too hard". I don't have anyone in my professional network or friend groups that knows anything about it, and information online is scarce. Learning from your course, and especially performing the exercises, has given me the **confidence to dive in** and start looking for bugs."

"The **processes and workflows** that you demonstrated. Particularly during your walkthroughs of the exercises, it was incredibly valuable to see and hear **your own methodology** for completing each example. The exercises themselves where also a fantastic learning tool."

"Here is what I loved about the whole thing:

- Well organized content, with a good order of things.
- A decent **balance of theory and hands-on** (I'm probably biased to hands-on).
- Pomodoro, time boxing, neural net.. liked the **meta-learning** touch there.
- Discussions on threat models, vuln discovery strategies, potential fuzzing designs."

"Loved the 25 minutes exercises, **really intense and gets you involved**."

Some public reviews from our students can be found in our Twitter favorites.

FURTHER INQUIRIES

E-mail: contact@zerodayengineering.com
Note: we typically respond to all business e-mails within 1-2 business days. If you didn't receive a response, that may be due to a failure of e-mail systems or spam filters.