Security and the Linux Kernel (LFD441)

Modality: Virtual Classroom

Duration: 4 Days

Intro to Course:

To learn the procedure and basic skeleton of Linux Kernel, the course is designed to focus on the very crucial tools suitable for debugging and keep an observatory eye the Kernel and how security features are deployed and commanded.

To the experienced programmers, this course offers strong comprehension of Linux Kernel, technical ideas of debugging and its tools. During 4 days course session, you will learn aggressive practical exercises and designed demonstrations to show very crucial tools to design and debug Linux Kernel Code.

An Embedded Linux Developer usually gets a pay of US $ 77,166 per annum on an average.

Who can enroll?

- App Developers
- C/C ++, C# Developers
- Linux Developers

Main Requirement!

Before you enrolled in this course, you must:

- Expert in C programming language.
- Have knowledge of basic Linux (UNIX) tools such as Is, Grep and Tar.
- Hand on any available text editors such as Emacs, Vi etc.
- An experience of important Linux distribution will be an added advantage, but not a requirement.
- Have an expertise equal to have taken LFD420, the Kernel Internals course.

Course Outline:

Introduction

- Objectives
- Who You Are
- The Linux Foundation{
- Copyright and No Confidential Information
- The Linux Foundation{ Training
- Certification Programs and Digital Badging
- Linux Distributions

- Platforms
- Things Change in Linux and Open Source Projects

Preliminaries

- Kernel Versions
- Kernel Sources and Use of git

Lab environment

- Virtual Machine
- Why proxmox {?
- Our Lab Environment
- Labs

How to Work in OSS Projects **

- Overview on How to Contribute Properly
- Know Where the Code is Coming From: DCO and CLA
- Stay Close to Mainline for Security and Quality
- Study and Understand the Project DNA
- Figure Out What Itch You Want to Scratch
- Identify Maintainers and Their Work Flows and Methods
- Get Early Input and Work in the Open
- Contribute Incremental Bits, Not Large Code Dumps
- Leave Your Ego at the Door: Don't Be Thin-Skinned
- Be Patient, Develop Long Term Relationships, Be Helpful

Reducing Attack Surfaces

- Why Security?
- Types of Security
- Vulnerabilities
- Layers of Protection
- Software Exploits
- Labs

Kernel Features

- Components of the Kernel
- User-Space vs. Kernel-Space
- What are System Calls?
- Available System Calls
- Scheduling Algorithms and Task Structures
- Process Context
- Labs

Kernel Deprecated Interfaces

- Why Deprecated
- __deprecated
- BUG() and BUG_ON()
- Computed Sizes for kmalloc()
- simple_strtol() Family of Routines
- strcpy(), strncpy(), strlcpy()
- printk() %p Format Specifier
- Variable Length Arrays
- Switch Case Fall-Through
- Zero-Length and One-Element Arrays in Structs

Address Space Layout Randomization (ASLR)

- Why ASLR?
- How to Use ASLR
- Disabling ASLR for Specific Programs
- Kernel Configuration
- Kernel Address Space Layout Randomization (KASLR)
- How KASLR Works
- Enabling KASLR
- Labs

Kernel Structure Layout Randomization

- Benefits
- How Structure Randomization Works
- Structure Initialization
- Opt-in vs Opt-out
- Partial Randomization
- Enabling Structure Randomization
- Building Out-of-tree Modules with Structure Randomization

Introduction to Linux Kernel Security

- Linux Kernel Security Basics
- Discretionary Access Control (DAC)
- POSIX ACLs
- POSIX Capabilities
- Namespaces
- Linux Security Modules (LSM)
- Netfilter
- Cryptographic Methods
- The Kernel Self Protection Project

CGroups

- Introduction to CGroups
- Overview

- Components of CGroup
- cgroup initialization
- cgroup Activation
- cgroups Parameters
- Testing cgroups
- systemd and cgroups
- Labs

eBPF

- BPF
- eBPF
- Installation
- bcc Tools
- bpftrace
- Labs

Seccomp

- What is seccomp
- The seccomp Interface
- seccomp Strict Mode
- seccomp Filter Mode
- Labs

Secure Boot

- Why Secure Boot?
- Secure Boot x86
- Embedded Systems Secure Boot
- Labs

Module Signing

- What is Module Signing?
- Basics of Signatures
- Module Signing Keys
- Enabling Module Signature Verification
- How It Works
- Signing Modules
- Labs

Integrity Measurement Architecture (IMA)

- Why IMA?
- Conceptual Operations
- Modes of Operation
- Collect Mode textit {(Collect and Store)

- Logging Mode textit {(Appraise and Audit)
- Enforcing Mode textit {(Appraise and Protect)
- Extended Verification Module (EVM)
- Labs

DM-Verity

- What is dm-verity?
- How dm-verity Works
- Enabling dm-verity
- Setting up dm-verity
- Using dm-verity
- Signing with dm-verity
- Booting with dm-verity
- Labs

Encrypted Storage

- Why Encrypted Storage?
- Data Encryption Solutions
- Survey of Storage Encryption Options
- Block Encryption
- Block Encryption Use
- Filesystem Encryption
- Filesystem Encryption Use
- Layered Filesystem Encryption
- Layered Filesystem Encryption Use
- Labs

Linux Security Modules (LSM)

- What are Linux Security Modules?
- LSM Basics
- LSM Choices
- How LSM Works
- An LSM Example: Yama
- Labs

SELinux

- SELinux
- SELinux Overview
- SELinux Modes
- SELinux Policies
- Context Utilities
- SELinux and Standard Command Line Tools
- SELinux Context Inheritance and Preservation**
- restorecon**

- semanage fcontext**
- Using SELinux Booleans**
- getsebool and setsebool**
- Troubleshooting Tools
- Labs

AppArmor

- What is AppArmor?
- Checking Status
- Modes and Profiles
- Profiles
- Utilities

Yama (LSM)

- Why Yama?
- Configuring Yama
- How Yama Works
- Labs

LoadPin (LSM)

- Why LoadPin?
- Enabling LoadPin
- Using LoadPin
- How LoadPin Works

Lockdown

- Why Lockdown?
- Lockdown Modes
- What Things are Locked Down?
- How It Works
- A Few Notes
- Labs

Safesetid

- Why Safesetid?
- Configuring Safesetid
- How Safesetid Works
- Labs

Netfilter

- What is netfilter?
- Netfilter Hooks

- Netfilter Implementation
- Hooking into Netfilter
- Iptables
- nftables
- Labs

Netlink Sockets**

- What are netlink Sockets?
- Opening a netlink Socket
- netlink Messages
- Labs

Closing and Evaluation Survey

- Evaluation Survey

Kernel Architecture I

- UNIX and Linux **
- Monolithic and Micro Kernels
- Object-Oriented Methods
- Main Kernel Components
- User-Space and Kernel-Space

Kernel Programming Preview

- Task Structure
- Memory Allocation
- Transferring Data between User and Kernel Spaces
- Object-Oriented Inheritance - Sort Of
- Linked Lists
- Jiffies
- Labs

Modules

- What are Modules?
- A Trivial Example
- Compiling Modules
- Modules vs Built-in
- Module Utilities
- Automatic Module Loading
- Module Usage Count
- Module Licensing
- Exporting Symbols
- Resolving Symbols **
- Labs

Kernel Architecture II

- Processes, Threads, and Tasks
- Kernel Preemption
- Real Time Preemption Patch
- Labs

Kernel Configuration and Compilation

- Installation and Layout of the Kernel Source
- Kernel Browsers
- Kernel Configuration Files
- Kernel Building and Makefiles
- initrd and initramfs
- Labs

Kernel Style and General Considerations

- Coding Style
- Using Generic Kernel Routines and Methods
- Making a Kernel Patch
- sparse
- Using likely() and unlikely()
- Writing Portable Code, CPU, 32/64-bit, Endianness
- Writing for SMP
- Writing for High Memory Systems
- Power Management
- Keeping Security in Mind
- Labs

Race Conditions and Synchronization Methods

- Concurrency and Synchronization Methods
- Atomic Operations
- Bit Operations
- Spinlocks
- Seqlocks
- Disabling Preemption
- Mutexes
- Semaphores
- Completion Functions
- Read-Copy-Update (RCU)
- Reference Counts
- Labs

Memory Addressing

- Virtual Memory Management

- Systems With and Without MMU and the TLB
- Memory Addresses
- High and Low Memory
- Memory Zones
- Special Device Nodes
- NUMA
- Paging
- Page Tables
- page structure
- Labs

Memory Allocation

- Requesting and Releasing Pages
- Buddy System
- Slabs and Cache Allocations
- Memory Pools
- kmalloc()
- vmalloc()
- Early Allocations and bootmem()
- Memory Defragmentation
- Labs