



SEC12 - Programmation de systèmes embarqués sécurisés

Objectifs

- Apprendre à vérifier que les programmes sont dans un état sécurisé au démarrage et lors des appels à d'autres programmes.
- Se familiariser avec les directives MISRA C pour l'utilisation du langage C dans les systèmes critiques
- Apprendre à utiliser le langage C/C++ en toute sécurité dans les systèmes critiques
- Apprendre à interpréter la sortie de l'outil de vérification MISRA C 2012
- comment manipuler les fichiers et les répertoires de manière sécurisée
- Découvrez comment protéger vos programmes contre les entrées malveillantes des utilisateurs
- Comment sécuriser la communication avec TLS
- Caractéristiques matérielles des systèmes embarqués pour la sécurité
- Méthodologie et cadre de développement de logiciels sécurisés
- Considération du logiciel du système sécurisé
- Appréhender le contexte et l'utilisation des hyperviseurs et de la virtualisation des systèmes
- Découvrir les contrôles et outils de sécurité

Pré-requis

- Quelques notions de programmation sont souhaitables (quel que soit le langage)

Environnement du cours

- Cours théorique
 - Support de cours au format PDF (en anglais)
 - Cours dispensé via le système de visioconférence Teams
 - Le formateur répond aux questions des stagiaires en direct pendant la formation et fournit une assistance technique et pédagogique
- Activités pratiques
 - Les activités pratiques représentent de 40% à 50% de la durée du cours
 - Elles permettent de valider ou compléter les connaissances acquises pendant le cours théorique.
 - Un PC Linux en ligne par stagiaire pour les activités pratiques
 - Exemples de code, exercices et solutions
 - Le formateur a accès aux PC en ligne des stagiaires pour l'assistance technique et pédagogique
 - Certains travaux pratiques peuvent être réalisés entre les sessions et sont vérifiés par le formateur lors de la session suivante
- Une machine virtuelle préconfigurée téléchargeable pour les activités pratiques après le cours
- Au début de chaque demi-journée une période est réservée à une interaction avec les stagiaires pour s'assurer que le cours répond à leurs attentes et l'adapter si nécessaire

Audience visée

- Tout ingénieur ou technicien en systèmes embarqués possédant les prérequis ci-dessus

Durée

- Totale : 30 heures
- 5 sessions de 6 heures chacune
- De 40% à 50% du temps de formation est consacré aux activités pratiques
- Certains travaux pratiques peuvent être réalisés entre les sessions et sont vérifiés par le formateur lors de la session suivante

Plan du cours

Première session

Introduction to embedded security

- Embedded Security Trends
 - Embedded Systems Complexity
 - Network connectivity
 - Reliance on Embedded Systems for Critical Infrastructure
 - Processor consolidation
- Security policies
 - Perfect Security
 - Confidentiality, Integrity, and Availability
 - Isolation
 - Information Flow Control
 - Physical Security Policies
 - Application-Specific Policies
- Security Threats

Writing Secure C/C++ Code

- Safe use of pointers
- Memory allocation and corruption
- Buffer overflow
- Return Oriented Programming
- Core embedded Operating system Security Requirements
- String and format functions
- Integer security
- Concurrency
- File I/O

Exercise : Memory Overflow Attacks

Deuxième session

Secure Coding

- Coding Standards
- Case Study: MISRA C:2012 and MISRA C++:2008
- Embedded C++
- Complexity Control
- Static Source Code Analysis
- Creating a Tailored Organizational Embedded Coding Standard
- Dynamic Code Analysis

Exercise : Use of static analysis tools

Cryptography Overview

- Cryptographic Modes
- Block Ciphers
- Authenticated Encryption
- Public Key Cryptography
- Key Agreement
- Public Key Authentication

- Elliptic Curve Cryptography
- Cryptographic Hashes
- Message Authentication Codes
- Random Number Generation
- Key Management for Embedded Systems

Exercise : Memory Overflow Attacks

Troisième session

Transport Layer Security

- Secure communications
- Authentication
- IoT Protocols
- MQTT
- DTLS
- HTTPS
- CoAP
- TLS Implementation
- Wireless LAN Security and Threats

Exercise : Installing and using certificates

Exercise : Sending secure messages with TLS

Secure Embedded System Software Architecture

- Secure software architecture goals
- Least privilege, trust and secure processes
- Arm Platform Security Architecture (PSA)

Secure Embedded System Hardware Architecture

- Crypto-Accelerator Overview
- Arm TrustZone
- Secure boot and update
- Hardware options for security

Quatrième session

System Software Consideration

- The Operating System
- Multiple Independent Levels of Security
 - Information Flow
 - Data Isolation
 - Damage Limitation
 - Periods Processing
 - Tamper Proof
 - Evaluable
- Core embedded Operating system Security Requirements
 - Memory Protection
 - Virtual Memory
- Guard Pages
- Location obfuscation
 - Fault Recovery
 - Impact of Determinism
 - Secure Scheduling
- Hypervisors and System Virtualization

- Introduction to System Virtualization
- Applications of System Virtualization
- Environment Sandboxing
- Virtual Security Appliances
- Hypervisor Architectures
- Paravirtualization
- Leveraging Hardware Assists for Virtualization
 - ARM TrustZone
- Hypervisor Security
- I/O Virtualization
- Remote Management
- Assuring Integrity of the TCB
 - Trusted Hardware and Supply Chain
 - Secure Boot
 - Static versus Dynamic Root of Trust
 - Remote Attestation

Exercise : Memory Protection (MPU)

Exercise : ARM TrustZone

Exercise : Secure Boot

Cinquième session

Data Protection Protocols for Embedded Systems

- Data-in-Motion Protocols
 - Generalized Model
 - Choosing the Network Layer for Security
 - Ethernet Security Protocols
 - IPsec versus SSL
 - IPsec
 - SSL/TLS
 - Embedded VPN Clients
 - DTLS
 - SSH
 - Custom Network Security Protocols
 - Secure Multimedia Protocols
 - Broadcast Security
- Data-at-Rest Protocols
 - Choosing the Storage Layer for Security
 - Symmetric Encryption Algorithm Selection
 - Managing the Storage Encryption Key

Testing for Security

- Basic Testing Methods
 - White-Box Testing
 - Black-Box Testing
 - Grey-Box Testing
- Fuzz-Testing