PROJECT REPORT OF INDUSTRY ORIENTED HANDS ON EXPERIENCE (IOHE) ON

Project at

LG Soft India

Submitted by

Nitin Jaswal

2110990970

Supervised By-

Praveen Kumar Reddy Yartha

Department of Computer Science and Engineering

CHITKARA UNIVERSITY
RAJPURA (PATIALA) PUNJAB-140401 (INDIA)

<u>CONTENTS</u>

Introduction

In this synopsis, I will share what I have learnt and implemented in the work I have done for LG Soft India, in the Cyber Security Department. Currently I am working on automotive cybersecurity and the module I am working on is the firewall of BMW. LG makes SOC [hardware] for different car brands and my project is on the firewall of a cars of BMW. The code is close to hardware, basically we are writing the custom OS for different SOC used in the car and the language used in the my module is C++ and some framework and libraries such as Binder IPC, Open SSL, GTest, and some other libraries that are private and are developed by the company.

Abstract

The main aim of my department is to create custom OS for various SOC's used in the car using the Yocto Build Systems, the team in which I am working deals with the management of firewall and its rules which prevents any third party device to enter or access the system and software of car.

Here are some key performed by me currently during my intern period [15 June 2023 - present].

- Understanding the basics of automotive cybersecurity and fixing the Build issues of my module.
- Understanding the basics of Open SSL and use them to encrypt and decrypt data.
- Understanding the IPC of the of the code and in particular of my project, Binder IPC framework is used to achieve the inter-process communication.
- Refactoring the code to reduce its complexity and migrating the project code to C++ 11.
- Writing the functional Unittests, for different APIs of the project using gtest framework in C++.

The application has a simple and interactive interface that allows the user to select an option from a menu and enter the required inputs. The application displays the outputs and errors in a clear and concise manner.

Methodology

The methodology I followed for my project was as follows:

 Research: In this step, I conducted extensive research on the theoretical aspects of cyber security concepts and terms from various reliable and authoritative sources, such as articles, video-tutorials, etc. I reviewed on topics such as encryption, decryption, certificates, signatures, CRLs, certificate chains, etc. I also learned about the OpenSSL library and its functions and features for implementing cyber security operations.

 Setup: In this step, I installed and configured the necessary tools and technologies for this project on my Windows system. I installed a C/C++ compiler that supports the latest standards and features of the language. I also installed VS code editor that provides a rich and user-friendly environment for coding and debugging C/C++ applications. I enabled WSL (Windows Subsystem for Linux) on my system that allows me to run Linux binaries and access Linux command-line tools, utilities, and applications without requiring a dual-boot or a virtual machine. I also installed OpenSSL library on WSL that provides various cryptographic functions and tools for generating keys, encrypting/decrypting data, creating/verifying certificates/signatures/CRLs, etc.

 Coding: In this step, I coded the C/C++ application using VS code editor in WSL environment. I followed the best practices and conventions of C/C++ programming and used appropriate algorithms, variables, functions, comments, etc. I also used standard C/C++ libraries for input/output, memory management, error handling, etc. I organized my code into separate files and modules for better readability and maintainability.

Testing: In this step, I tested and debugged the application using various inputs and outputs. I used VS code editor’s built-in debugging tools to identify and fix any errors or bugs in my code. I also verified the validity and correctness of the results. I ensured that the application meets the functional and non-functional requirements and performs as expected.

Tools and Technologies

The tools and technologies used for this project are:

- C/C++: C/C++ is a versatile programming language that provides low-level control over hardware and memory. It is widely used for system programming and high-performance applications. It is suitable for coding cyber security operations because it supports bitwise operations, pointers, structures, etc. that are essential for encryption, decryption, hashing, etc.
- VS code: VS code is a code editor that supports multiple programming languages and platforms. It provides features such as syntax highlighting, code completion, debugging, version control, extensions, etc. that improve the productivity and efficiency of coding. It is convenient for coding C/C++ applications because it has built-in support for C/C++ language and debugging tools. It also allows integration with WSL environment that enables cross-platform development.
- WSL: WSL is a compatibility layer that allows running Linux binaries on Windows systems. It provides access to Linux command-line tools, utilities, and applications without requiring a dual-boot or a virtual machine. It is useful for coding C/C++ applications that use OpenSSL library because it provides a Linux-like environment that supports OpenSSL installation and execution.
- gtest: Google Test (gtest) is an open-source framework for unit testing C++ projects.

  It's based on the xUnit architecture, which is a set of frameworks for programming and automated execution of test cases.
- Binder: Binder is an inter-process communication (IPC) mechanism in Android. It

  allows one Android process to call a routine in another Android process. Binder identifies the method to invoke and passes the arguments between processes.

OpenSSL : OpenSSL is an open-source library that implements SSL/TLS protocols and provides cryptographic functions. It provides various tools and functions for generating keys, encrypting/decrypting data, creating/verifying certificates/signatures/CRLs, etc. It is ideal for coding cyber security operations because it supports various algorithms, standards, formats, etc. that are widely used in cyber security domain.

MACSEC

•MKAD (MKA Daemon) is a user-space daemon responsible for implementing the MACsec Key Agreement (MKA) protocol, as defined in the IEEE 802.1X-2010 standard. MKA is a crucial component of MACsec (Media Access Control Security), which provides secure communication at Layer 2 of the OSI model by encrypting and authenticating Ethernet frames. MKAD facilitates the dynamic exchange and management of cryptographic keys between MACsec-capable peers using the EAPOL-MKA (Extensible Authentication Protocol over LAN - MACsec Key Agreement) protocol. It establishes and maintains Security Associations (SAs), negotiates Secure Channels (SCs), and distributes session keys required for MACsec encryption and decryption.

In our project, MKAD was integrated into an embedded Linux environment to enable end-to-end secure communication between two systems over a physical Ethernet link. We configured and validated MKAD to perform secure peer authentication, real-time key negotiation, and automated secure channel setup. Additionally, we verified the encrypted communication using packet captures and confirmed the integrity and confidentiality of the transmitted data. MKAD’s integration demonstrates a practical and secure implementation of Layer 2 encryption, making it highly suitable for securing communication in automotive, industrial, and enterprise embedded systems

Kernel Hardening

•Kernel hardening enhances the security of the Linux kernel by minimizing vulnerabilities and protecting against exploitation. Since the kernel operates with full system privileges, securing it is critical. One primary approach is reducing the attack surface by disabling unused features, drivers, and filesystems during kernel configuration. Security features like Kernel Address Space Layout Randomization (KASLR) and Stack Canaries protect against memory-based attacks by randomizing memory addresses and detecting buffer overflows. These measures, along with continuous updates and patches, create a layered defense that makes it significantly harder for attackers to exploit kernel-level vulnerabilities, especially in servers and embedded systems.