Agenda

- Safety vs Security
- Why Check Secure Coding Guidelines?
- How to Apply Secure Coding Guidelines?
Safety vs. Security

Note: Security issues may cause safety issues
5W1H - Secure Coding Guidelines

Who: You or Your colleagues (Developers, QA, QE)
Where: at workplace
What: Most of your software running on target
When: Every day, week, month
5W1H - Secure Coding Guidelines

**Why**
1. Security issues may lead to catastrophe
2. Required by your customers

**How**
1. Use static analysis tool for security
2. Make analysis jobs automated
Why Check Secure Coding Guidelines?
Cybersecurity – Emerging Topic in the Auto Industry

- Growing communication of on-board systems, sensors and external sites
- Car becomes another node of IoT
- Security of automotive embedded systems increasingly important (possible cyber attacks)

FCA recalls 1.4 Million cars after Jeep hack

https://youtu.be/MK0SrxBC1xs
Security in Consumers’ Mind

- 57% of customers think automakers and suppliers are responsible for protecting data.

- 91% of customers would never buy or wary buying from automakers were hacked.

Source: 2019 KPMG Consumer Loss Barometer study.
Typical Embedded Software Architecture

Interaction with external components?

Control Algorithm, Fault Detection, Supervisory Logic

Utility (I/O Driver, Lookup Table, etc.)

RTOS, Fault Logging, Service Tool Interface
Embedded Software External Interactions

- Control Algorithm, Fault Detection, Supervisory Logic
- Utility (I/O Driver, Lookup Table, etc.)
- RTOS, Fault Logging, Service Tool Interface
- Network
- File System
- HSM
- 3rd party software
- User Input
- Sensors

MathWorks
AUTOMOTIVE CONFERENCE 2019

10
Embedded Software Security Concerns

- Incorrect order of network connection operations
- Tainted Data
- TOCTOU
- Vulnerable path manipulation
- Use of non-secure temporary file
- Deterministic random output from constant seed
- Vulnerable pseudo-random number generator
- Sensitive heap memory not cleared before release
- Execution of a binary/Load of library from a relative path can be controlled by an external actor
- Tainted Data
Let’s look at an example - Tainted Data

```c
#include <stdio.h>
#define ADCMAXSIZE 256

typedef signed int sint32;
typedef unsigned char uint8;

extern sint32 getLengthRxData(void);
extern sint32 readByte(void);

void receiveData(void)
{
    sint32 i, length;
    sint32 ADCdata[ADCMAXSIZE];

    length = getLengthRxData();

    for (i = 0; i < length; i++)
    {
        ADCdata[i] = readByte();
    }
}
```
Let’s look at an example - Tainted Data

- One of the most exploited and extremely high risk vulnerability – buffer overflow!

```c
#include <stdio.h>
define ADCMAXSIZE 256
typedef signed int sint32;
typedef unsigned char uint8;
extern sint32 getLengthRxData (void);
extern sint32 readByte (void);
void receiveData(void)
{
    sint32 i, length;
sint32 ADCdata[ADCMAXSIZE];
    length = getLengthRxData();
    for (i = 0; i < length; i++)
    {
        ADCdata[i] = readByte();
    }
}
```

Polyspace helps you find those vulnerabilities
How to Apply Secure Coding Guidelines?
Most Frequently Heard Secure Coding or Security Standards

- CERT C/C++
- ISO/IEC 17961
- MISRA-C:2012 Amendment 1, Addendum 2/3
- CWE (Common weakness enumeration)
# Embedded Safety and Security Coding Standards Overview

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MISRA C:2004</td>
<td>C89</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MISRA C:2012</td>
<td>C90/99</td>
<td>Yes*</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CERT C99</td>
<td>C99</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CERT C11</td>
<td>C11</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ISO/IEC TS 17961</td>
<td>C11</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>CWE</td>
<td>None/all</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

*Additional security guidelines for MISRA-C:2012 Amendment 1

Source: Table is based on the book: [The CERT C Coding Standard](https://www.amazon.com/CERT-C-Coding-Standard-Reliable/dp/1449399661)
How does *Polyspace* help you with embedded software security?

- Detecting security vulnerabilities and underlying defects early
- Provides Exhaustive Documentation and recommendation for security fix
- Proving absence of certain critical vulnerabilities
- Complying with industry standards such as CERT C/C++ and ISO 17961
Polyspace Tools

Bug Finder
- Produce code metrics
- Check coding standards
- Find defects and vulnerabilities

Code Prover
- Proves code Safe and Secure
- 33 most critical run-time checks
- Supports DO-178 and ISO 26262
Easy to Configure and Review

- Provides explicit options to check Security Guidelines like CERT-C and ISO 17961
- Enable/Disable each rules easily
- Checks Security, Safety guidelines and Defects in One tool
Important to close all the windows!

- False negatives - missed vulnerabilities
- All malicious attackers want is one loop hole
- Testing is not exhaustive, almost all static analysis tools are not exhaustive
- Polyspace Code Prover – proving absence of specific vulnerabilities
- For critical defects such as buffer overflows, illegal pointer dereferencing..
Let’s look at an example - Tainted Data

```c
#include <stdio.h>
#define ADCMAXSIZE 256

typedef signed int sint32;
typedef unsigned char uint8;

extern sint32 getLengthRxData (void);
extern sint32 readByte (void);

void receiveData(void)
{
    sint32 i, length;
    sint32 ADCdata[ADCMAXSIZE];

    length = getLengthRxData();

    for (i = 0; i < length; i++)
    {
        ADCdata[i] = readByte();
    }
}
```
Polyspace helps you find those vulnerabilities

```c
#include <stdio.h>
define ADCMAXSIZE 256
typedef signed int sint32;
typedef unsigned char uint8;

extern sint32 getLengthRxData(void);
extern sint32 readByte(void);

void receiveData(void)
{
    sint32 i, length;
    sint32 ADCdata[ADCMaxSize];

    length = getLengthRxData();

    for (i = 0; i < length; i++)
    {
        ADCdata[i] = readByte();
    }
}
```

*Warning: array index may be outside bounds: [0..255]*
array size: 256
array index value: [0..256]
Polyspace helps you find those vulnerabilities
Recommended Workflow

**Pre-Submit Workflows**
- Desktop or IDE Plugin
  - Polyspace Bug Finder
- Desktop or IDE Plugin
  - Polyspace Bug Finder
- Desktop or IDE Plugin
  - Polyspace Code Prover

**Post-Submit Workflows**
- Build Engineer
  - Build automation tool (e.g., Jenkins)
  - Polyspace Bug Finder Server
  - Polyspace Code Prover Server

**Analysis Automation**
- Server 1: Runs Analysis

**Collaborative Review**
- Developer
- Team Lead
- QA/QE
- Web Browser

**Recommended Workflow**
- Developer
  - Desktop or IDE Plugin
    - Polyspace Bug Finder
  - Server 2: Hosts Results
  - Polyspace BF Access
  - Polyspace CP Access
Automate Polyspace Analysis by Jenkins plug-in

- Set env variables for Polyspace Access and Web Metrics
- Ease analysis automation and configuration for a standard CI workflow

Create project automatically
Launch an analysis
Upload analysis result
Generate a report
e-mail to owners
Polyspace Access - Review Results in Collaborative Environment
Key takeaways

# 1. Achieve the Goals for Security and Safety by One Tool, Polyspace

# 2. Improve your workflow by Analysis Automation & Collaborative Review Environment
Thank you!