정형 기법을 활용한 AUTOSAR SWC의 구현 확인 및 정적 분석

Develop high quality embedded software

이 영준
Principal Application Engineer
Agendas

- Unit-proving of AUTOSAR Component and Runtime error

- Secure Coding Standard and Polyspace
  - MISRA-C:2012 Amendment 1
  - ISO 17961
  - CERT-C/C++
  - CWE
Unit-Proving of AUTOSAR Component And Runtime Error
What is AUTOSAR?

- The Automotive industry and its challenges

OEMs objectives:
- Integration from different suppliers
- Need confidence in the supplier’s code

Supplier’s challenge:
- Time-to-market
- Code size
- Pressure from OEMs

AUTOSAR solves by providing a software architecture and common specifications (ARXML files)

Need for validation of AUTOSAR components among actors
How Polyspace for AUTOSAR can help?

ARXML files are used to communicate, Polyspace for AUTOSAR is used to prove robustness and compliance.

1) check for run-time errors and mismatch in the ARXML specifications
2) Check if implementation follow specifications
3) assess impact of changes in the specifications
4) check implementation against specifications updates
Polyspace for AUTOSAR features

Automatic split by component

Polyspace

Sound analysis

Sound analysis plus checks to prove that the code matches the specification

ARXML

Implementation (source files)

Automatic launching on each component

Back to specifications

New view to detail the AUTOSAR specification

Prove specs matching
Polyspace for AUTOSAR workflow

Specifications
- ARXML files
  - SW-C Description
  - AUTOSAR

Split the code in components as defined in ARXML

Polyspace for AUTOSAR
- Performance a separate unit analysis of each component with Polyspace
- Free of run-time errors
- Checks that code of runnable respects its output specification
- Checks that code of runnable calls Rte functions in respect of their specification

Implementation
- Simulink model
- swcA.c
- swcA.h
- ...
- swcB.c
- Simulink model
  - Project browser
  - Configuration
  - Options
    - Target & Checkpoints
      - Macros
      - Environment Settings
      - Inputs & Outputs
      - Coding Styles & Code Metrics
      - Bug Finder Analysis
      - Code-Profiler Verification
      - Verification Assumptions
      - Check Behavior
      - Profile
      - Scaling
      - Report
      - Run Settings
      - Advanced Settings
- Configuration
  - Compiler
  - Preprocessor
  - Linker
  - Linker
  - Library
  - Library
  - Source
  - Target
  - Project
  - Cmake
  - Make
  - Make
Polyspace and AUTOSAR

AUTOSAR architecture

Application Layer

RTE

Services Layer

ECU Abstraction Layer

Complex Device Drivers

Microcontroller Abstraction Layer

ECU Hardware

ARXML provides specification of Application Layer and link with RTE

Polyspace verifies the match between code and ARXML

Polyspace verifies the Application Layer

Polyspace stubs the RTE Layer

RTE Layer not verified by “Polyspace for AUTOSAR”

Polyspace can verify RTE

Not verified by “Polyspace for AUTOSAR”

Polyspace may verify these
Unit verification of an AUTOSAR software component

Component = set of runnable functions

Verifies each runnable in isolation

Checks that code of runnable respects its output specification

ARXML provides specification of runnables:
- Context of call
- Output

ARXML provides specification of Rte functions

Calls the runnables as defined in specification

C code

Checks that code of runnable calls Rte functions in respect of their specification
Unit verification of an AUTOSAR software component

Simulink model

ARXML

ARXML provides specification of runnables:
- Context of call
- Output

Runnables

C code

Checks that code of runnable respects its output specification

Checks that code of runnable calls Rte functions in respect of their specification

Service / ECU Layers
Hand-Written Code based on ARXML

Polyspace for AUTOSAR SWC
Generated Code From Simulink Model based on ARXML

Polyspace for AUTOSAR SWC
Workflow Benefits

- Provide **automatically** the best configuration for Polyspace

- Detect **inconsistencies** between AUTOSAR specifications and code implementation

- **Unit** verification of AUTOSAR software components with Polyspace
  - *Sound* analysis: proves that code respects the specification
  - *Static* analysis: considers all potential cases
Secure Coding and Polyspace
Safety vs. Security

Note: Security issues may cause safety issues
Cybersecurity – Industry Activities & Standards

SAE – Vehicle Cybersecurity Systems Engineering Committee

- SAE J3061 - Cybersecurity Guidebook for Cyber-Physical Vehicle Systems
- SAE J3101 - Requirements for Hardware-Protected Security for Ground Vehicle Applications (WIP)
- SAE “Cybersecurity Assurance Testing Task Force” (TEVEES18A1)

Coding standards & practices that we observe at automotive customers

- MISRA-C:2012 Amendment 1
- ISO/IEC TS 17961 – C Secure Coding Rules
- CERT-C / CERT-C++
- CWE – Common Weakness Enumeration
## ISO/IEC TS 17961 Compared with Other Standards

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

Table is based on the book:
This coding standard consists of **rules** and **recommendations**, collectively referred to as **guidelines**.

- **Rules** are meant to provide normative requirements for code, whereas

- **Recommendations** are meant to provide guidance that, when followed, should improve the safety, reliability, and security of software systems.
CERT-C Coverage with Polyspace

- You can map Polyspace results to CERT C rules and recommendations
- Using Polyspace results, you can address 103 CERT C rules (90%) and 95 CERT-C recommendations (50%)
  - The CERT C website, under continuous development, lists 118 rules and 188 recommendations (Count based on The CERT C++ Coding Standard document, 2016 Edition)
CERT-C++ coverage with Polyspace

- You can map Polyspace results to CERT C++ rules
- Using the Polyspace results, you can address 34 CERT C++ rules (40%) and 79 CERT C rules that also apply to C++ (99%)

- The [CERT C++ website](https://www.cert.org/certcplus), under continuous development, lists 163 rules including 80 CERT C rules that also apply to C++ (based on count in April 2018 in CERT-C++ website)

✓ Two new arguments for option `-checkers` in C++ mode (-lang CPP): `CERT-rules` (only CERT-C++ rules) and `CERT-all` (it includes also CERT-C rules that apply)
Completeness And Soundness

*From ISO 17961*

- **False Negatives**
  - Failure to report a real flaw in the code is usually regarded as the most serious analysis error, as it may leave the user with a false sense of security.

- **False Positives**
  - The tool reports a flaw when one does not exist.

### Table 1 — Completeness and soundness

<table>
<thead>
<tr>
<th>False negatives</th>
<th>False positives</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>N</td>
<td>Sound with false positives</td>
<td>Complete and sound</td>
</tr>
<tr>
<td>Y</td>
<td>Unsound with false positives</td>
<td>Complete and unsound</td>
</tr>
</tbody>
</table>

Polyspace Code Prover

Polyspace Bug Finder
ISO/IEC TS 17961
C secure coding rules
ISO 17961 - C Secure Coding Rules

- The purpose of this Technical Specification is to specify analyzable secure coding rules that can be automatically enforced to detect security flaws in C-conforming applications.

- To be considered a security flaw, a software bug must be triggerable by the actions of a malicious user or attacker.
ISO 17961 - C Secure Coding Rules

3.2 Coverage Summary

<table>
<thead>
<tr>
<th>Classification</th>
<th>Strength</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit</td>
<td>Strong</td>
<td>20</td>
</tr>
<tr>
<td>Implicit</td>
<td>Strong</td>
<td>2</td>
</tr>
<tr>
<td>Restrictive</td>
<td>Strong</td>
<td>5</td>
</tr>
<tr>
<td>Partial/Restrictive</td>
<td>Strong/None</td>
<td>2</td>
</tr>
<tr>
<td>None</td>
<td>None</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>46</td>
</tr>
</tbody>
</table>
Polyspace Bug Finder And Security Standard

- Well-know defects for unreliable code like buffer overflows, dead code...
- Plus two categories: Security and Tainted data

- Security Standards
  - CERT-C
  - ISO-17961 (Full)
  - MISRA-C 2012 (Full)
  - CWE

- The mapping table between Polyspace Bug Finder and Security Standard
  - MATLAB_INSTALL\polyspace\resources\Polyspace Results R2018b.xlsx
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 – MISRA-C, CWE, CERT C, ISO 17961
Q & A