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

Specifications

OEMs (manufacturers)
- HMC
- BMW
- ...

ARXML files

Tier-1 (suppliers)
- Hyundai ......
- Bosch
- ...

Implementations

Set up the supply chain

Components providers (general and need to be calibrated to a specific vehicle)

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?

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

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

Automatic split by component

Sound analysis

Back to specifications

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

New view to detail the AUTOSAR specification

Automatic launching on each component

Prove specs matching

ARXML

Implementation (source files)

polyspace-autosar

SwC1

SwC2

SwC3

Polyspace

swcA.c
swcA.h
...
swcB.c

Automatic launching on each component
Polyspace for AUTOSAR workflow

Specifications

- ARXML files

Split the code in components as defined in ARXML

Polyspace for AUTOSAR

Perform 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

Specifications

- ARXML files

Implementation

- Simulink model

- swcA.c
- swcA.h
- ...
- swcB.c

✓ 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
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

C code

Checks that code of runnable respects its output specification

Verifies each runnable in isolation

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

ARXML

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

ARXML provides specification of Rte functions

 Calls the runnables as defined in specification

Rte functions

Service / ECU Layers
Unit verification of an AUTOSAR software component

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

ARXML provides specification of Rte functions

Simulink model

Calls the runnables as defined in specification

Checks that code of runnable respects its output specification

Rte functions

Service / ECU Layers

C code

Checks that code of runnable calls Rte functions in respect of their specification
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:
SEI CERT C Coding Standard

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

**Table 1 — Completeness and soundness**
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

5.1 Accessing an object through a pointer to an incompatible type [ptrcomp] .............. 5
5.2 Accessing freed memory [acfree] ........................................................................ 6
5.3 Accessing shared objects in signal handlers [accsig] ........................................... 7
5.4 No assignment in conditional expressions [boolasgn] ........................................... 8
5.5 Calling functions in the C Standard Library other than abort, _Exit, and signal from within a signal handler [asyncsig] ................................................................. 9
5.6 Calling functions with incorrect arguments [argcomp] ......................................... 11
5.7 Calling signal handlers .......................................................................................... 12
5.8 Calling system functions ....................................................................................... 13
5.9 Comparison of error codes .................................................................................... 14
5.10 Converting a printf or scanf argument .................................................................. 15
5.11 Converting pointers ............................................................................................. 16
5.12 Copying a FILE ...................................................................................................... 17
5.13 Declaring the same type ....................................................................................... 18
5.14 Dereferencing a NULL pointer .............................................................................. 19
5.15 Escaping the NULL character ............................................................................... 20
5.16 Conversion of size_t to EOF [signconv] ................................................................. 21
5.17 Use of an implicit conversion ................................................................................. 22
5.18 Falling to close files ............................................................................................. 23
5.19 Falling to detect error ........................................................................................... 24

5.20 Forming invalid pointers by library function [libpnt] ........................................... 25
5.21 Allocating insufficient memory [insufmem] ......................................................... 26
5.22 Forming or using out-of-bounds pointers or array subscripts [invptr] ................. 27
5.23 Freeing memory multiple times [dblefree] ........................................................... 28
5.24 Including tainted or out-of-domain input in a format string [usrsfmt] ............... 29
5.25 Incorrectly setting and using errno [inverrno] ..................................................... 30
5.26 Integer division errors [diverr] .............................................................................. 31
5.27 Interleaving stream inputs and outputs without a flush or positioning call ........ 32
5.28 Modifying string literals [strmod] ....................................................................... 33
5.29 Modifying the string returned by getenv, localeconv, setlocale, and strftime [libmod] ........................................................................................................... 34
5.30 Overflowing signed integers [intflow] ................................................................... 35
5.31 Passing a non-null terminated character sequence to a library function that expects a string [nonnull] ................................................................. 36
5.32 Passing arguments ............................................................................................... 37
5.33 Passing pointers in function argument parameters [restpar] ............................... 38
5.34 Reallocation or freeing of pointers ....................................................................... 39
5.35 Referencing uninitialized storage ......................................................................... 40
5.36 Subtracting or comparing floating point values ................................................... 41
5.37 Tainted strings are used in a context that requires a non-tainted string ................ 42
5.38 Taking the size of a pointer to determine the size of the pointed-to type [sizeofpnt] ............................................................................................................. 43
5.39 Using a tainted value as an argument to an unprototyped function [taintproto] ............................................................................................................ 44
5.40 Using a tainted value to write to an object using a formatted input or output function [unformatio] .............................................................. 45
5.41 Using a value for isetpos other than a value returned from fsetpos [filepos] ...... 46
5.42 Using an object overwritten by getenv, localeconv, setlocale, and strftime [libuse] .......................................................................................................... 47
5.43 Using character values that are indistinguishable from EOF [cheroe] ................. 48
5.44 Using identifiers that are reserved for the implementation [resident] ................. 49
5.45 Using invalid format strings [invalidstr] .............................................................. 50
5.46 Tainted, potentially mutilated, or out-of-domain integer values are used in a restricted sink [taintsink] ...................................................................................... 51

3.2 Coverage Summary

In summary, the coverage of MISRA C:2012 against C Secure is as follows:

<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>1</td>
</tr>
<tr>
<td>Restrictive</td>
<td>Strong</td>
<td>11</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><strong>Total</strong></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