Kudelski IoT Security

KTA Code Check

Kudelski Trust Agent

CC Restricted - Kudelski Confidential
15.08.2024

Copyright notice

DOCUMENT PROPERTIES

Version	0.0.1
File name	KTA CODE CHECK.DOCX
Publication date	15.08.2024
Confidentiality Status	CC Restricted - Kudelski Confidential
Document owner	Kudelski Group
Document recipient	
Document status	Draft
Client company name	KSIOT

TABLE OF CONTENTS

TABLE OF FIGURES

No table of figures entries found.

TABLE OF TABLES

No table of figures entries found.

1. EXECUTIVE SUMMARY

This document contains the code analysis report of the Kudelski Trust Agent (KTA), specified in document [R1].

It describes the static code analysis methodology and describes how the various code checks are performed regarding:

- Coding rules compliance
- Static code analysis
- MISRA C:2012 compliance

2. METHODOLOGY

The code static analysis is performed with the Cppcheck v2.10 static analysis tool. Cppcheck is an open-source static code analysis tool for C/C++ code (see [R3]).

The Cppcheck tool is run four times with different parameters to:

- Enforce Kudelski coding rules
- Perform static code checks (pointers, overflows…)
- Ensure compliance with MISRA C:2012 rules, MISRA C:2012 Amendment 1 & MISRA C:2012 Amendment 2
- Ensure compliance with CERT rules

The following Cppcheck options are common to all configurations:

```
--platform=win32A
--std=c99
--enable=warning,style,performance,portability
--template=gcc
--inconclusive --inline-suppr
--suppress=missingIncludeSystem
--suppress=redundantAssignment
--suppress=variableScope
--suppress= unreadVariable
--suppress=misra-c2012-17.7
--suppress=misra-c2012-21.6
--suppress=misra-c2012-15.1
--suppress=misra-c2012-15.4
--suppress=misra-c2012-21.10
--suppress=misra-c2012-2.5
--suppress=misra-c2012-20.10
--suppress=misra-c2012-5.9
--suppress=misra-c2012-8.7
--cppcheck-build-dir=./MisraReports
--addon=./DEV/TOOLS/PROJECT/MISRACHECK/misra.json
--platform=native
--suppress=*:*/EMU/*
```

Generic suppressions:

missingIncludeSystem is suppressed as we assume includes are available.

redundantAssignment is suppressed due to security feature. Returned status is always initialized with an error value.

variableScope is suppressed as all local variables are declared in the beginning of the functions.

unreadVariable

The code static check is also run using the Sonarqube static analysis tool. The Sonarqube tool comprises of two parts –

- Build wrapper – This tool is used to compile the project as per sonarqube specification and create a json file which contains the information of compiled files for analysis.

- Sonar Scanner – This is a CLI tool to run the static check analysis and upload the details to the sonarqube server (hosted by organization).

2.1. Executing the Code Check

The full code check is started by executing the following command in the /DEV/SOURCE/kta and in the /DEV/SOURCE/salapi folder.:

Each check can be started individually:

- MISRA checks

```
cppcheck -D <kta_feature_flag> -D <pre_processor_symbols> -I <header_files> --
<supressions> "list_of_c_files"
```

Note that certcheck shall be executed only on computer having access to the CppCheck premium license.

3. CODING RULES

In order to enforce the coding rules defined in **Error! Reference source not found.**, Cppcheck is run to generate the source code *Abstract Syntax Tree* (AST). The AST is then processed by an in-house Python script (codecheck.py) which verifies that the coding rules are followed.

3.1. CPPCHECK OPTIONS

Cppcheck is run with the default options listed in §2.

Note: the KTA header files are not part of the verification as they were delivered to the customer before the coding rules were enforced. Updating the header files would change the API and impact the customer.

3.2. CPPCHECK RESULT

No error.

4. STATIC CODE ANALYSIS

On its own, Cppcheck analyses the source code and looks for undefined behavior that may lead to bugs or vulnerabilities, such as:

- Dead pointers
- Division by zero
- Integer overflows
- Invalid bit shift operands
- Invalid conversions
- Invalid usage of STL
- Memory management
- Null pointer dereferences
- Out of bounds checking
- Uninitialized variables
- Writing const data

4.1. CPPCHECK OPTIONS

Cppcheck is run with the default options listed in §2.

4.2. CPPCHECK RESULT

No error.

5. MISRA DIRECTIVES

Directive	Category	Status	Comment
1.1	Required	Compliant	KTA makes no use of standard library, no assembly code, no use of division nor floating point operation
2.1	Required	Deviation	KTA code is compiled without errors and with as few warning as possible See C Compilers chapter
3.1	Required	Compliant	Code is fully traceable to documented requirements
4.1	Required	Compliant	MISRA Rule 1.3 is verified
4.2	Required	Compliant	No use of assembly code
4.3	Required	Compliant	No use of assembly code
4.4	Required	Compliant	Use of #ifdef instead of commented code
4.5	Advisory	Disapplied	Identifiers follow the project’s coding rules
4.6	Advisory	Compliant	Code uses <stdint.h> types. Size_t is used in interface
4.7	Required	Deviation	Return codes are always tested
4.8	Advisory	Compliant	MISRA Rule 17.7 is verified
4.9	Advisory	Disapplied	Function-like-macros are present only in logging code. This code is not present in final product
4.10	Required	Compliant	.h files format follow the coding rules
4.11	Required	Compliant	No library is used except for logging that is not present in final product
4.12	Required	Compliant	No dynamic memory allocation
4.13	Advisory	Compliant	No access to resources

AMENDMENT 1 DIRECTIVE:

Directive	Category	Status	Comment
4.14	Required	Compliant	Inputs are checked before being used

5.1. DEVIATIONS FROM "REQUIRED" DIRECTIVES

The KTA implementation deviates from the following MISRA C 2012 "Required" directives.

MISRA Directive #	Synopsis	Category
2.1	All source files shall compile without any compilation errors	project

MISRA Directive #	Synopsis	Category
4.7	If a function returns error information, then that error information shall be tested	project

For directive 2.1, please refer to §6 for C compilers options.

Deviation of directive 4.7 is accepted as it implements a security best practice. This practice consists in wiping sensitive buffers in case of error. As it is an error management, this kind of code cannot return an error and shall silently ignore wiping errors.

6. MISRA C:2012 COMPLIANCE

This chapter identifies the MISRA C 2012 (see [R1]) non-compliances ("Advisory" rules) and deviations ("Required" rules).

The non-compliance/deviation category is:

- "Project" when the rule is disabled for the whole project
- “file” when the rule is disabled on a specific source file
- "Specific" when the rule is disabled on a case-by-case basis in source files

6.1. CPPCHECK COMPLIANCE

Cppcheck is able to check the following MISRA C:2012 guidelines.

Guideline	Cppcheck	Category	Status
1.1		Required	Compliant
1.2		Advisory	Compliant
1.3	X	Required	Compliant
2.1	X	Required	Compliant
2.2	X	Required	Compliant
2.3	X	Advisory	Compliant
2.4	X	Advisory	Compliant
2.5	X	Advisory	Deviation
2.6	X	Advisory	Compliant
2.7	X	Advisory	Deviation
3.1	X	Required	Compliant
3.2	X	Required	Compliant
4.1	X	Required	Compliant
4.2	X	Advisory	Compliant
5.1	X	Required	Compliant
5.2	X	Required	Compliant
5.3	X	Required	Compliant
5.4	X	Required	Compliant
5.5	X	Required	Compliant
5.6	X	Required	Compliant
5.7	X	Required	Compliant
5.8	X	Required	Compliant
5.9	X	Advisory	Deviation
6.1	X	Required	Compliant
6.2	X	Required	Compliant
7.1	X	Required	Compliant
7.2	X	Required	Compliant
7.3	X	Required	Compliant
7.4	X	Required	Compliant
8.1	X	Required	Compliant
8.2	X	Required	Compliant
8.3	X	Required	Compliant
8.4	X	Required	Compliant
8.5	X	Required	Compliant
8.6	X	Required	Compliant
8.7	X	Advisory	Deviation
8.8	X	Required	Compliant
8.9	X	Advisory	Compliant
8.10	X	Required	Compliant
8.11	X	Advisory	Compliant
8.12	X	Required	Compliant
8.13	X	Advisory	Compliant
8.14	X	Required	Compliant
9.1	X	Mandatory	Compliant
9.2	X	Required	Compliant
9.3	X	Required	Compliant
9.4	X	Required	Compliant
9.5	X	Required	Compliant
10.1	X	Required	Compliant
10.2	X	Required	Compliant
10.3	X	Required	Compliant
10.4	X	Required	Compliant
10.5	X	Advisory	Compliant
10.6	X	Required	Compliant
10.7	X	Required	Compliant
10.8	X	Required	Compliant
11.1	X	Required	Compliant
11.2	X	Required	Compliant
11.3	X	Required	Compliant
11.4	X	Advisory	Compliant
11.5	X	Advisory	Compliant
11.6	X	Required	Compliant
11.7	X	Required	Compliant
11.8	X	Required	Compliant
11.9	X	Required	Compliant
12.1	X	Advisory	Compliant
12.2	X	Required	Compliant
12.3	X	Advisory	Compliant

Guideline	Cppcheck	Category	Status
12.4	X	Advisory	Compliant
13.1	X	Required	Compliant
13.2	X	Required	Compliant
13.3	X	Advisory	Compliant
13.4	X	Advisory	Compliant
13.5	X	Required	Compliant
13.6	X	Mandatory	Compliant
14.1	X	Required	Compliant
14.2	X	Required	Compliant
14.3	X	Required	Compliant
14.4	X	Required	Compliant
15.1	X	Advisory	Deviation
15.2	X	Required	Compliant
15.3	X	Required	Compliant
15.4	X	Advisory	Deviation
15.5	X	Advisory	Compliant
15.6	X	Required	Compliant
15.7	X	Required	Compliant
16.1	X	Required	Compliant
16.2	X	Required	Compliant
16.3	X	Required	Compliant
16.4	X	Required	Compliant
16.5	X	Required	Compliant
16.6	X	Required	Compliant
16.7	X	Required	Compliant
17.1	X	Required	Deviation
17.2	X	Required	Compliant
17.3		Mandatory	Compliant
17.4	X	Mandatory	Compliant
17.5	X	Advisory	Compliant
17.6	X	Mandatory	Compliant
17.7	X	Required	Deviation
17.8	X	Advisory	Compliant

Guideline	Cppcheck	Category	Status
18.1	X	Required	Compliant
18.2	X	Required	Compliant
18.3	X	Required	Compliant
18.4	X	Advisory	Compliant
18.5	X	Advisory	Compliant
18.6	X	Required	Compliant
18.7	X	Required	Compliant
18.8	X	Required	Compliant
19.1	X	Mandatory	Compliant
19.2	X	Advisory	Compliant
20.1	X	Advisory	Compliant
20.2	X	Required	Compliant
20.3	X	Required	Compliant
20.4	X	Required	Compliant
20.5	X	Advisory	Compliant
20.6	X	Required	Compliant
20.7	X	Required	Compliant
20.8	X	Required	Compliant
20.9	X	Required	Compliant
20.10	X	Advisory	Compliant
20.11	X	Required	Compliant
20.12	X	Required	Compliant
20.13	X	Required	Compliant
20.14	X	Required	Compliant
21.1	X	Required	Compliant
21.2	X	Required	Compliant
21.3	X	Required	Compliant
21.4	X	Required	Compliant
21.5	X	Required	Compliant
21.6	X	Required	Deviation
21.7	X	Required	Compliant
21.8	X	Required	Compliant
21.9	X	Required	Compliant

Guideline	Cppcheck	Category	Status
21.10	X	Required	Deviation
21.11	X	Required	Compliant
21.12	X	Advisory	Compliant
22.1	X	Required	Compliant
22.2	X	Mandatory	Compliant
22.3	X	Required	Compliant
22.4	X	Mandatory	Compliant
22.5	X	Mandatory	Compliant
22.6	X	Mandatory	Compliant

Amendment 1 Guidelines:

Guideline	Cppcheck	Category	Status
12.5	X	Mandatory	Compliant
21.13	X	Mandatory	Compliant
21.14	X	Required	Compliant
21.15	X	Required	Compliant
21.16	X	Required	Compliant
21.17	X	Mandatory	Compliant
21.18	X	Mandatory	Compliant
21.19	X	Mandatory	Compliant
21.20	X	Mandatory	Compliant
22.7	X	Required	Compliant
22.8	X	Required	Compliant
22.9	X	Required	Compliant
22.10	X	Required	Compliant

Amendment 2 Guideline:

Guideline	Cppcheck	Category	Status
21.21	X	Required	Compliant

Code is compiled with both GCC (version 11.3) and MSVC (v142x86) in order to verify rules 1.1 & 1.2.

Rule 1.1: *The program shall not contain no violations of the standard C syntax and constraints and shall not exceed the implementation's translation limits*

Rule 1.2: *Language extensions should not be used*

6.2. Cppcheck Options

MISRA C:2012 rules are enforced with Cppcheck using the following options, in addition to those listed in §2:

```
--suppress=misra-c2012-17.7
--suppress=misra-c2012-21.6
--suppress=misra-c2012-15.1
--suppress=misra-c2012-15.4
--suppress=misra-c2012-21.10
--suppress=misra-c2012-2.5
--suppress=misra-c2012-20.10
--suppress=misra-c2012-5.9
--suppress=misra-c2012-8.7

 --addon=DEV/TOOLS/PROJECT/MISRACHECK/misra.json
```

misra.json

```
{
  "script": "misra.py",
  "args": [
    "--rule-texts=../../../TOOLS/PROJECT/MISRA_C2012_Rules.txt"
  ]
}
```

6.3. DEVIATIONS FROM "REQUIRED" RULES

The KTA implementation deviates from the following MISRA C 2012 "Required" rules.

MISRA Rule #	Synopsis	Category
17.1	The standard header file shall not be used.	Project
17.7	The value returned by a function having non-void return type shall be used.	Project
21.6	The Standard Library input/output functions shall not be used.	Project
21.10	The Standard Library time and date functions shall not be used.	Log

See the corresponding deviation records in §**Error! Reference source not found.**

6.4. NON-COMPLIANCES WITH "ADVISORY" RULES

The KTA implementation does not comply with the following MISRA C 2012 "Advisory" rules.

MISRA Rule #	Synopsis	Category
15.1	The goto statement should not be used.	Project
15.4	There should be no more than one break or goto statement used to terminate any iteration statement.	Project

As no deviation process is required for "advisory" rules, they are just listed here.

6.5. Related deviation permit

R-17.1_01:

Rule 17.1 The features of <stdarg.h> shall not be used

Permit / KSIOT / C:2012 / R-17.1_01

Exceptional use of stdargs is accepted

Reason Performance efficiency (Resource utilization, small code footprint)

Background

As code footprint can be critical in some projects, the feature can be used.

Requirements

- All instance of code using stdarg shall be identified in documentation

- Maximum (minimum) number of parameters shall be explicitly tested before parameters being used.

- Parameters type (or their value ranges) shall be checked before these parameters being used

6.6. Deviation Report.

6.6.1. MISRA_DEV_KTA_001

Project	KTA
Deviation ID	MISRA_DEV_KTA_001
MISRA C Reference	17.1 – The standard header file shall not be used.
Permit	N/A
Status	Analysis
Reason	Usability (Accessibility)
Tracing tags	MISRA_DEV_KTA_001
Source	KTA Library
Scope	Project

6.6.1.1. Summary

The purpose of this rule is to avoid the use of standard header files that might introduce code that is non-compliant with the MISRA guidelines. Standard header files typically define standard library functions and macros which might not be fully compliant with MISRA's rules, leading to unsafe or undefined behavior

This means that instead of using standard headers like <stdio.h>, <stdlib.h>, <string.h>, etc.,

6.6.1.2. Detailed Description

The KTA defines *C_KTA_APP__LOG* to print the debug and error logs. The define is using the stdio function printf.

```
#define C_KTA_APP__LOG printf
```

6.6.1.3. Justification

As KTA uses logging for errors, it uses the stdio functionality to print the logs to the terminal.

6.6.1.4. Conditions under which the deviation is requested

This deviation is applied overall KTA source code.

6.6.1.5. Consequences on non-compliance

6.6.1.6. Actions to control reporting

In *misraCheck.sh*, the following Cppcheck command line parameter is used to suppress the error in the whole project:

```
MISRA_OPT+="--suppress=misra-c2012-17.1 "
```

6.6.2. MISRA_DEV_KTA_002

Project	KTA
Deviation ID	MISRA_DEV_KTA_002
MISRA C Reference	17.7 – The value returned by a function having non-void return type shall be used.
Permit	N/A
Status	Analysis
Source	KTA Library
Reason	Usability (Accessibility)
Tracing tags	MISRA_DEV_KTA_002
Scope	Project

6.6.2.1. *Summary*

This rule is designed to ensure that the return value of a function with a non-void return type is always used. Ignoring the return value of such functions can lead to unintended behavior, bugs, or missed errors, particularly in embedded systems where reliability is critical.

6.6.2.2. *Detailed Description*

The KTA uses functions like snprintf, memcpy, memset without using the return values.

These functions are non-void functions.

```
snprintf(aBuffer, C_MAX_BUFFER_SIZE, "%s : [%d]\r\n", xpFmt, xSize);
```

6.6.2.3. *Justification*

The KTA uses the function snprintf to compose the logs throughout the project. The usage of this functionality is purely for logging purpose and pose no harm on the production and release variant.

6.6.2.4. *Conditions under which the deviation is requested*

This deviation is applied on the whole KTA source code.

6.6.2.5. *Consequences on non-compliance*

6.6.2.6. *Actions to control reporting*

In *misraCheck.sh*, the following Cppcheck command line parameter is used to suppress the error in the whole project:

```
MISRA_OPT+="--suppress=misra-c2012-17.7 "
```

6.6.3. MISRA_DEV_KTA_003

Project	KTA
Deviation ID	MISRA_DEV_KTA_003
MISRA C Reference	21.6 – The Standard Library input/output functions shall not be used.
Permit	N/A
Status	Analysis
Source	KTA Library
Reason	Usability (Accessibility)
Tracing tags	MISRA_DEV_KTA_003
Scope	Project

6.6.3.1. Summary

The project needs to include standard io headers

6.6.3.2. Detailed Description

In this project for example k_sal_log.c there is a usage of the function salPrint, which calls the stdio function printf for displaying debug and error logs.

One of the example is mentioned below.

```
void salPrint(char* buffer)
{
    printf("%s", buffer);
}
```

6.6.3.3. Justification

The project uses logging mechanism to display logs for certain cases. These logging mechanism uses printf functionality of the stdio.h library.

6.6.3.4. Conditions under which the deviation is requested

This deviation is applied on the whole KTA source code.

6.6.3.5. Consequences on non-compliance

There are no consequences as there is no risk of loss of precision.

6.6.3.6. Actions to control reporting

In *misraCheck.sh*, the following Cppcheck command line parameter is used to suppress the error in the whole project:

```
MISRA_OPT+="--suppress=misra-c2012-21.6 "
```

6.6.4. MISRA_DEV_KTA_004

Project	KTA
Deviation ID	MISRA_DEV_KTA_004
MISRA C Reference	15.1 – The goto statement should not be used.
Permit	N/A
Status	Analysis
Source	KTA Library
Reason	Usability (Accessibility)
Tracing tags	MISRA_DEV_KTA_004
Scope	Project

6.6.4.1. Summary

MISRA Rule 15.1 prohibits the usage of goto and jump statements.

6.6.4.2. Detailed Description

In the project there are usage of goto statements to reduce the number of nesting functions by using multiple if…else statements and also to prevent the usage of multiple return statements in case of null check errors and similar errors in a function.

6.6.4.3. Justification

Removing the goto statements will lead to usage of multiple return or increase the nesting steps in a function, where there is a need to return midway through the function due to some error occurrence.

6.6.4.4. Conditions under which the deviation is requested

This deviation is applied on the whole KTA source code.

6.6.4.5. Consequences on non-compliance

There are no consequences as there is no risk of loss of precision. Also there is no risk to its success or integrity.

6.6.4.6. Actions to control reporting

In *misraCheck.sh*, the following Cppcheck command line parameter is used to suppress the error in the whole project:

```
MISRA_OPT+="--suppress=misra-c2012-15.1 "
```

6.6.5. MISRA_DEV_KTA_005

Project	KTA
Deviation ID	MISRA_DEV_KTA_005
MISRA C Reference	15.4 – There should be no more than one break or goto statement used to terminate any iteration statement.
Permit	N/A
Status	Analysis
Source	KTA Library
Reason	Usability (Accessibility)
Tracing tags	MISRA_DEV_KTA_005
Scope	Project

6.6.5.1. Summary

MISRA Rule 15.4 prohibits the usage of multiple goto and jump statements.

6.6.5.2. Detailed Description

In the project there are usage of goto statements to reduce the number of nesting functions by using multiple if…else statements and also to prevent the usage of multiple return statements in case of null check errors and similar errors in a function.

6.6.5.3. Justification

Removing the goto statements will lead to usage of multiple return or increase the nesting steps in a function, where there is a need to return midway through the function due to some error occurrence.

6.6.5.4. Conditions under which the deviation is requested

This deviation is applied on the whole KTA source code.

6.6.5.5. Consequences on non-compliance

There are no consequences as there is no risk of loss of precision. Also there is no risk to its success or integrity.

6.6.5.6. Actions to control reporting

In *misraCheck.sh*, the following Cppcheck command line parameter is used to suppress the error in the whole project:

```
MISRA_OPT+="--suppress=misra-c2012-15.4 "
```

6.6.6 MISRA_DEV_KTA_006

Project	KTA
Deviation ID	MISRA_DEV_KTA_006
MISRA C Reference	21.10 – The Standard Library time and date functions shall not be used.
Permit	N/A
Status	Analysis
Source	KTA Library
Reason	Usability (Accessibility)
Tracing tags	MISRA_DEV_KTA_006
Scope	Project

6.6.6.1 Summary

MISRA Rule 15.4 prohibits the usage of time and date library

6.6.6.2 Detailed Description

In the project there are the usage of time and date library for capturing the log time.

6.6.6.4 Justification

Removing the timestamp for logging will reduce the efficiency of the logs

6.6.6.5 Conditions under which the deviation is requested

This deviation is applied on the whole KTA source code.

6.6.6.6 Consequences on non-compliance

There are no consequences as there is no risk of loss of precision. Also there is no risk to its success or integrity.

6.6.6.7 Actions to control reporting

In *misraCheck.sh*, the following Cppcheck command line parameter is used to suppress the error in the whole project:

```
MISRA_OPT+="--suppress=misra-c2012-21.10 "
```

6.6.7 MISRA_DEV_KTA_007

Project	KTA
Deviation ID	MISRA_DEV_KTA_007
MISRA C Reference	2.5 – A macro should be used at least once in the code.
Permit	N/A
Status	Analysis
Source	KTA Library
Reason	Usability (Accessibility)
Tracing tags	MISRA_DEV_KTA_007
Scope	Project

6.6.7.1 Summary

MISRA Rule 2.7 prohibits defining macros that are never used. Some macros are intentionally defined but conditionally unused in certain builds.

6.6.7.2 Detailed Description

Macros may be left unused in specific configurations or reserved for future features.

6.6.7.3 Justification

Macros are retained for configuration consistency and do not impact compiled output. Removing them would complicate code alignment between variants.

6.6.7.4 Conditions under which the deviation is requested

Allowed only if clearly documented as reserved, optional, or build specific.

6.6.7.5 Consequences on non-compliance

Risk of confusion over unused macros and possible accumulation of dead definitions.

6.6.7.6 Actions to control reporting

In *misraCheck.sh*, the following Cppcheck command line parameter is used to suppress the error in the whole project:

```
MISRA_OPT+="--suppress=misra-c2012-2.5"
```

6.6.8 MISRA_DEV_KTA_08

Project	KTA
Deviation ID	MISRA_DEV_KTA_08
MISRA C Reference	20.10 – The preprocessor operators should not be used.
Permit	N/A
Status	Analysis
Source	KTA Library
Reason	Usability (Accessibility)
Tracing tags	MISRA_DEV_KTA_007
Scope	Project

6.6.8.1 Summary

MISRA Rule 20.10 discourages the preprocessor stringizing (#) and token-pasting (##) operators because they can reduce code clarity and make debugging harder

6.6.8.2 Detailed Description

When used in a controlled and well-documented manner (e.g., for consistent macro expansion or reducing repetitive code), these operators improve maintainability and do not cause unintended side effects.

6.6.8.3 Justification

In rare, controlled cases, the lifetime of the referenced object is guaranteed (e.g., pointer only used immediately before returning), so no dangling reference occurs.

6.6.8.4 Conditions under which the deviation is requested

Allowed only in low-level headers or framework code where alternatives would significantly increase duplication.

6.6.8.5 Consequences on non-compliance

Strictly avoiding # and ## may lead to verbose, repetitive code, increasing maintenance effort and chance of human error.

6.6.8.6 Actions to control reporting

In *misraCheck.sh*, the following cppcheck command line parameter is used to suppress the error in the whole project

```
MISRA_OPT+="--suppress=misra-c2012-20.10"
```

6.6.9 MISRA_DEV_KTA_009

Project	KTA
Deviation ID	MISRA_DEV_KTA_009
MISRA C Reference	5.9 – Identifiers that define objects or functions with external linkage shall be unique across translation units
Permit	N/A
Status	Analysis
Source	KTA Library
Reason	Usability (Accessibility)
Tracing tags	MISRA_DEV_KTA_007
Scope	Project

6.6.9.1 Summary

MISRA Rule 5.9 ensures that all external identifiers (global variables, functions with external linkage) are unique to prevent name collisions during linking.

6.6.9.2 Detailed Description

If two different source files define the same external name, behavior is undefined or may lead to linker errors. In some cases, legacy code or third-party libraries may use common names that conflict but are intentionally isolated in build configuration.

6.6.9.3 Justification

Deviation is permitted only when the build system guarantees that conflicting definitions are never linked together (e.g., mutually exclusive modules or conditionally compiled code).

6.6.9.4 Conditions under which the deviation is requested

If not controlled, name collisions can result in link-time errors or unintended symbol resolution causing runtime faults.

6.6.9.5 Consequences on non-compliance

Prefer using static linkage where possible to avoid exposing symbols.

6.6.9.6 Actions to control reporting

In *misraCheck.sh*, the following cppcheck command line parameter is used to suppress the error in the whole project

```
MISRA_OPT+="--suppress=misra-c2012-5.9"
```

6.6.010 MISRA_DEV_KTA_010

Project	KTA
Deviation ID	MISRA_DEV_KTA_010
MISRA C Reference	8.7 – Functions and objects should not be defined with external linkage if they are referenced in only one translation unit
Permit	N/A
Status	Analysis
Source	KTA Log File
Reason	Usability (Accessibility)
Tracing tags	MISRA_DEV_KTA_007
Scope	Project

6.6.10.1 Summary

MISRA Rule 8.7 requires that objects (variables) with internal linkage shall be declared static. This prevents unintentional exposure of variables outside the translation unit and enforces encapsulation.

6.6.10.2 Detailed Description

A non-static declaration without extern defaults to external linkage, even if the intention is to restrict visibility within a single file. Failing to mark such variables as static can cause namespace pollution and possible conflicts with similarly named identifiers in other translation units.

6.6.10.3 Justification

Deviation is permitted when a variable is intentionally designed for external linkage and shared across multiple modules, when legacy or third-party code depends on global variables without static for compatibility, or when the build system and coding guidelines enforce strict naming conventions that prevent symbol collisions.

6.6.10.4 Conditions under which the deviation is requested

A variable must be accessed by multiple source files and cannot be restricted to file scope. Refactoring to use static or accessor functions is not feasible due to performance or architectural constraints

6.6.10.5 Consequences on non-compliance

If not controlled, non-compliance may lead to unintended access or modification of variables by other modules, an increased risk of namespace conflicts and linkage errors, and reduced maintainability with unclear module boundaries.

6.6.10.6 *Actions to control reporting*

In *misraCheck.sh*, the following cppcheck command line parameter is used to suppress the error in the whole project

```
MISRA_OPT+="--suppress=misra-c2012-8.7"
```

6 C COMPILERS

6.6 MPLABX GNUWIN32 C COMPILER

KTA Code is compiled with MPLAB v6.20 GNUWin32 C compiler.

6.7 CYGWIN GCC

KTA Code has been compiled with GCC version 11.3. The following relevant options have been used:

- -std=c99 (use C99 standard)
- -Wall (enable all warnings)
- -Wextra (enable extra warnings)
- -Werror (treat warning as error)
- -Wno-error=sign-compare

This compilation generates one warning “comparison of promoted bitwise complement of an unsigned value with unsigned”. This is due to a GCC bug (see GCC Bugzilla #38341).

DOCUMENT HISTORY

Version Status	Date	Author / Changes
0.0.1 **Draft**	2024-05-29	KTA Team Leader

Reviewer	Position	Date	Document Version

Approver	Position	Date	Document Version
KTA Team Leader	Team leader	2024-05-29	0.0.1

DOCUMENT RECIPIENTS

Name	Position	Contact Info

KUDELSKI IOT CONTACT

Name	Position	Contact Info
www.kudelski-iot.com		

REFERENCES

[R1] MISRA C:2012 Guidelines for the use of the C language in critical systems – March 2013
[R2] SEI CERT C Coding Standard – 2016 Edition
[R3] Cppcheck static analysis tool - https://cppcheck.sourceforge.io/
[R4] MISRA C:2012 Amendment 1 – April 2016
[R5] MISRA C:2012 Amendment 2 – February 2020

ACRONYMS

Acronym	Stands For
KTA	Kudelski Trust Agent for KSE5

APPENDICES