Best Programming Practices Using MISRA-C and its Compliance

Ashwina Kumar & Dr. Bikash Kanti Sarkar
Dept. of computer science and engg. Birla Institute of Technology, Mesra, Ranchi, Jharkhand, India

Received: March 09, 2019 Accepted: April 19, 2019

ABSTRACT: : *MISRA C is a set of software development guidelines for the C programminglanguage developed by MISRA (Motor Industry Software Reliability Association). Since their original use in the automotive industry, they havebeen accepted worldwide, as the benchmark for C coding standards acrossall safety sectors where safety, quality or reliability are issues of concern.*
Its aims are to facilitate code safety, security, portability, and reliability inthe context of embedded systems.Compliance to MISRA-C is integral part of the process for getting certificationin Automotive industry. The project aims for making the bootloadercode MISRA-C compliant.

Key Words: *MISRA-C , Motor Industry*

I. Introduction

To promote best practices in developing safety and security related electronicand embedded systems and other software-intensive application.MISRA-C is a set of software development guidelines for the C programminglanguage developed by MISRA (Motor Industry Software ReliabilityAssociation).MISRA stands for **The Motor Industry Software Reliability Association**, The organization produces guidelines to create Safe and Reliable softwarefor automotive industry. MISRA-C is the guidelines for coding in Clanguage. These reduce or remove opportunity to make mistakes in safetycritical systems like automobiles. MISRA-C++ is the guidelines for C++coding. **Functional safety standard ISO-26262 certification for automotiveindustry mandates software to be MISRA compliant.**In this report we will read about the MISRA-C and how it is used incompliance. First we will discuss about the why MISRA-C in chapter2. In chapter 3 and 4, we will discuss about MISRA-C Compliance and deviationsalong with MISRA-C tools.

II. METHODOLOGY

Coverity tool is used to get the MISRA-C report for the project. It is a static analyzer tool which gives the count of violations for the different rules which are being build. It consists of 5 steps.

A. Steps for coverity set-up to get MISRA report

These are the following setps to get the MISRA-C report for any project.

1. Clean Build: This is the step in which we must build the project so that there is no compilation issue.
2. Cov-build: This is the step in which emit the project which got compiled to be analyzed.
3. Cov-analyze: This is the step in which emitted project got analyzed using some specified MISRA config and the count of violations get grouped.
4. Cov-report: This provides the MISRA report by specifying the line number along with the rule number in the code tree.
5. Commit to database: This will commit the defects/violations to coverity connect database.

III. PROPOSED METHOD

There a mechanism by which we can get the MISRA report for any sample C code. The code should be isolated and must not contains any dependency. For any dependent code we have some different way of generating the report. Here, we would discuss about isolated and independent code.

Example: take the below sample code. Now we need to find out the MISRA violations for this test code. For this we would run a script and find out the violations.

```
#include<stdint.h>
int main(void)
{
  int32_t num = 2;
  uint32_t temp = num;
  return 0;
}
```

Violations report

```
1 #include<stdint.h>
2
3 int main(void)
4 {
5 int32_t num = 2;
```

(1) Event misra_c_2012_rule_10_3_violation: Implicit conversion of "num" from essential type "signed 32-bit int" to different or narrower essential type "unsigned 32-bit int".

```
6 uint32_t temp = num;
7 return 0;
8 }
9
```

```
1 #include<stdint.h>
2
3 int main(void)
4 {
5 int32_t num = 2;
```

(1) Event misra_c_2012_rule_2_2_violation: variable "temp" was declared but never referenced
(2) Event caretline: ^

```
6 uint32_t temp = num;
7 return 0;
8 }
9
```

A. RESULTS

The above sample C code is taken for reference and we have tried to find out the violations in that code.

1. Install the coveirty provided by Synopsys by doing all the steps of installations.
2.Login: Give the credentials in the like username and password for login.
3. Run: We need to run the report using some command line script and then we can see the snapshot for that run. It would give us the number of violations which are fixed along with the newly generated violations.

Figure 1: Cov-analysis step

```
Output path: /tmp/cov_out
[WARNING] Template config template-gcc-config-0 already exists for gcc and will be reused.
[WARNING] Template config template-g++-config-0 already exists for g++ and will be reused.
[WARNING] Template config template-gcc-config-1 already exists for gcc-3 and will be reused.
[WARNING] Template config template-gcc-config-2 already exists for gcc-4 and will be reused.
[WARNING] Template config template-g++-config-1 already exists for g++-3 and will be reused.
[WARNING] Template config template-g++-config-2 already exists for g++-4 and will be reused.
[WARNING] Template config template-ld-config-0 already exists for ld and will be reused.
```

```
Emitted 1 C/C++ compilation units (100%) successfully

1 C/C++ compilation units (100%) are ready for analysis
The cov-build utility completed successfully.
Coverity Static Analysis version 2018.12 on Linux 4.4.0-131-generic x86_64
Internal version numbers: c56d2f65de p-pacific-push-35824

Using 12 workers as limited by CPU(s)
Looking for translation units
|0----------25----------50----------75---------100|
**************************************************
[WARNING] SECURE_CODING is deprecated. Use the Don't-Call checkers (DC.*) instead.
[STATUS] Computing links for 1 translation unit
|0----------25----------50----------75---------100|
**************************************************
[STATUS] Computing function pointers
|0----------25----------50----------75---------100|
**************************************************
[STATUS] Computing virtual overrides
|0----------25----------50----------75---------100|
**************************************************
[STATUS] Computing callgraph
|0----------25----------50----------75---------100|
**************************************************
[STATUS] Topologically sorting 1 function
|0----------25----------50----------75---------100|
**************************************************
[STATUS] Resolving dataflow directives
|0----------25----------50----------75---------100|

Analysis summary report:
------------------------
Files analyzed : 1
Total LoC input to cov-analyze : 781
Functions analyzed : 1
Paths analyzed : 1
Time taken by analysis : 00:00:03
Defects/Coding rule violations found : 2 Total
1 MISRA C-2012 Rule 10.3
1 MISRA C-2012 Rule 2.2

Processing 2 errors.

0% 10 20 30 40 50 60 70 80 90 100%
|----|----|----|----|----|----|----|----|----|----|
**************************************************
Processed errors.
```

Figure 4: Login enters the mail id and password and click on next button.

B. Pros and Cons:

The pros of coverity are:

1. The user can see the violations in GUI form.
2. It enhances the concept of re usability.
3. It saves a lot of time.
4. Easy to write scripts because of the tabular formats.

The cons of keyword-driven testing are [6,11]:

1. Sometimes it gave the wrong violations which is eventually false positive and that needs to be handled carefully.
2. It gives the high turnaround time when the projects are big and contains several projects.
3. The complexity of the server increases proportionally with the numberofprojects.

Conclusion & Future scope

MISRA-C is a set of software development guidelines for the C programminglanguage developed by MISRA (Motor Industry

Software Reliability Association).The static code analysis is used to get the report of MISRA-C violationsper rule. It tells thenumber of violations per Rule in the form of CID's.There is some work which could be done in future to solve the problem ofMISRA-C violations fix and its Coverity database.Coverity version 2018.12 set-up should be done in future keeping in mindsome rules that are wrongly pointed in earlier coverity versions.

References

[1] https://www.misra.org.uk/MISRAHome/WhatisMISRA/tabid/66/Default.aspx
[2] https://www.embedded.com/electronics-blogs/beginner-scorner/4023981/Introduction-to-MISRA-C
[3] https://en.wikipedia.org/wiki/Coverity
[4] https://ldra.com/aerospace-defence/standards/misra-cc/
[5] http://www.cosmicsoftware.com/misra.php
[6] https://www.perforce.com/resources/qac/misra-c-cpp
[7] https://misra.org.uk/LinkClick.aspx?fileticket=V2wsZxtVGkE
[8] https://gitlab.com/MISRA/MISRA-C/MISRA-C-2012/Example-Suite
[9] https://www.gimpel.com/html/misra.htm
[10] https://www.embedded.com/electronics-blogs/beginner-scorner/4023981/Introduction-to MISRA-C
[11] https://www.techopedia.com/definition/32099/automation
[12] https://docs.python.org/2/library/parser.html
[13] https://en.wikipedia.org/wiki/MISRA_C
[14] https://www.researchgate.net/publication/316859493_MISRA_C_for_Security's_Sake
[15] https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/MISRA_C.h ml
[16] https://www.bugseng.com/effective-misra-c
[17] https://arxiv.org/abs/1809.00821
[18] https://sourceforge.net/p/cppcheck/discussion/general/thread/ccbe9e89/
[19] https://wiki.sei.cmu.edu/confluence/display/c/MISRA+C%3A2012
[20] http://frey.notk.org/books/MISRA-Cpp-2008.pdf
[21] https://www.slideshare.net/AdaCore/misra-c-recent-developments-and-a-road-map-to-the-future