NUREG/CR-6463 가이드라인에 대한 Cppcheck 정적분석 규칙 집합 개발

송승현○ 정세진 유준범
건국대학교 컴퓨터공학부
union93@konkuk.ac.kr, jsjj0728@gmail.com, jbyoo@konkuk.ac.kr

Index

Introduction

Introduction

원자력 발전소에서 안전성의 중요성

안전성은 원자력 발전소와 같은 실시간 임베디드 시스템에서 중요한 요소이다. NRC(U.S. Nuclear Regulatory Commission)와 같은 원자력 규제 기관에서는 NUREG-0800을 통해 SRP(Standard Review Plan)을 제공하여 소프트웨어 개발주기 동안 안전성 검사를 실행하도록 권장 하고있다.

SRP에 포함된 코드 리뷰 과정과 NUREG/CR-6463

NUREG-0800은 BTP 7-14파트에서 NUREG/CR-6463 혹은 NUREG/CR-6463을 포함하는 코딩 가이드라인을 사용하여 원자력 발전소에 사용되는 소프트웨어를 코드 리뷰 할 것을 권장하고 있다.

Introduction

기존의 NUREG/CR-6463 코드 가이드라인 활용방법

NUREG/CR-6463은 코드 리뷰 가이드라인으로서 코드 리뷰 시 체크리스트화 하거나 코딩 컨벤션 지침으로 사용되었다. 그러나 이러한 방식의 활용은 원자력 발전소와 같이 큰 규모의 시스템을 분석하는데 비효율적이며, 개발자의 역량에 따라 검사 결과가 달라질 수 있다.

정적분석 도구에서 활용가능한 NUREG/CR-6463 기반의 규칙 개발 필요

본 연구에서는 정적분석 도구 Cppcheck에서 사용가능한 NUREG/CR-6463에 기반한 규칙집합을 개발하였다. 이를 통해 정적분석을 통해 C코드로 작성 된 소프트웨어의 NUREG/CR-6463 위반사항을 효율적이고 보다 일관성 있게 검출 가능하도록 하였다. 개발된 규칙집합이 위반사항을 검출하는지 확인하기 위하여 RPS(Reactor Protection System)에 사용되는 로직을 구현한 C프로그램을 Cppcheck로 정적분석 하였다.

NUREG/CR-6463

NUREG/CR-6463

NRC에서 제안한 코드 리뷰 가이드라인으로 원자력 발전소 소프트웨어 개발에 사용되는 High-Level Language의 코드 리뷰 가이드라인을 제시한다.

NUREG/CR-6463은 Reliability, Robustness, Traceability, Maintainability라는 4가지 소프트웨어 안전성 속성을 중심으로 작성 되어있다.

속성	설명
Reliability	소프트웨어가 성공적으로 실행될 가능성과 요구되는 컨디션 동안 성공적으로 작동할 가능성에 관여하는 속성
Robustness	소프트웨어가 비정상적인 조건이나 이벤트에서 허용가능한 방식으로 작동할 가능성에 관한 속성
Traceability	소스 코드 및 라이브러리 구성요소 출처 및 개발 프로세스를 검토하고 식별하는 것과 관련이 있는 속성
Maintainability	소프트웨어의 유지보수에 관한 속성

NUREG/CR-6463 기반의 Cppcheck ruleset 개발

자연어로 기술된 NUREG/CR-6463

NUREG/CR-6463은 자연어로 작성된 가이드라인으로 정적분석에 사용하기 어렵다. Cppcheck에서 제공하는 Custom Rule 형태로 개발하기 위해서는 Pattern과 Message를 추출하여 XML 형태로 작성되어야 한다.

Pattern

소스코드에서 검출할 위반사항과 일치하는 문자열을 POSIX 정규표현식으로 나타낸 것

Message

Pattern과 일치하는 소스코드가 발견될 경우 사용자에게 보여지는 정보

NUREG/CR-6463 기반의 Cppcheck ruleset 개발

표1 NUREG/CR-6463의 원문의 규칙 개발 결과 예시

NUREG/CR-6463 Guideline	Code Sample	Original Text in NUREG/CR-6463	Proposed Rule for Cppcheck
4.1.2.2 Minimizing Control Flow Complexity	O	Use the switch construct. In safety systems, the switch ... case construct should be used to replace multiple if ... else if ... else if ... statements if possible (Porter, 1993). In the example below, test-value is the only term used for evaluation.	<rule version="1"> <pattern>\belse if\b</pattern> <message> <id>M.16</id> <severity>error</severity> <summary>Avoid use of the ?: operator</summary> </message> </rule>
4.4.1.6 Minimizing Obscure or Subtle Programming Construct	X	Avoid use of the ?: operator. The ?: operator is another form of the if-then-else statement. The ? : operator makes the code more difficult to read should be avoided in favor of the more conventional if-then-else construct.	<rule version="1"> <pattern>[][?:]</pattern> <message> <id>M.16</id> <severity>error</severity> <summary>Avoid use of the ?: operator</summary> </message> </rule>

NUREG/CR-6463에서 샘플을 제공하는 경우

샘플코드가 제공되는 경우 샘플 코드를 Cppcheck를 통해 토큰정보를 추출

NUREG/CR-6463에서 샘플을 제공하지 않는 경우

가이드라인에서 명시하고 있는 변수 타입, 함수 이름, 스타일을 토큰정보로 추출

일부 가이드라인의 제외

NUREG/CR-6463 가이드라인 중 일부는 정적분석이 불가능 하다. 예를 들어 4.3.2 Document all cases of dynamic binding to externally developed libraries 가이드라인은 문서작성에 관한 가이드라인으로 정적분석도구로 확인하는 것이 불가능하다. 이러한 가이드라인들은 개발에서 제외되었기 때문에 전체 104개의 NUREG/CR-6463의 가이드라인 중 29개의 가이드라인을 제외하고 75개의 가이드라인이 정적분석 가능한 규칙 집합으로 개발되었다.

NUREG/CR-6463 기반의 Cppcheck ruleset 개발

C and C++ SECTION in NUREG/CR-6463		Number of guideline	Number of rulset
4.1 Reliability			
4.1.1	Predictability of Memory Utilization	9	8
4.1.2	Predictability of Control Flow	45	36
4.1.3	Predictability of Timing	3	2
4.2 Robustness			
4.2.1	Controlled Use of Software Diversity	0	0
4.2.2	Controlled Use of Exception Handling	7	7
4.2.3	Input and Output Checking	1	1
4.3 Traceability			
4.3.1	Minimizing the Use of Built-In Functions	1	1
4.3.2	Minimizing the Use of Compiled Libraries	3	3
4.3.3	Utilizing Version Control Tools	0	0
4.4 Maintainability			
4.4.1	Readability	21	9
4.4.2	Data Abstraction	6	3
4.4.3	Functional Cohesiveness	0	0
4.4.4	Malleability	0	0
4.4.5	Portability	8	5
Total		**104**	**75**

표2 NUREG/CR-6463 섹션 별 가이드라인 개수와 개발된 규칙의 개수

Case Study

Case Study

개발된 규칙이 NUREG/CR-6463 가이드라인 위반을 검출 가능한지 확인

RPS에서 사용되는 FBD(Function Block Diagram)으로 표현된 5가지 로직 fixed set-point falling trip (FFT), fixed set-point rising trip (FRT), manual reset falling trip (MFT), variable set-point falling trip (VFT), and variable set-point rising trip (VRT)[8]을 NUDE 2.0(Nuclear Development Environment)의 FBDtoC 기능을 사용하여 C프로그램으로 변환하여 정적분석 하였다

검사 환경

OS: Ubuntu LTS 18.04 버전
Static Analysis tool: Cppcheck 1.82
Static Analysis Rule: Cppcheck 1.82 support

Case Study

Line	Id	CWE	Severity	Message
Header_FBD.h				
24	M.29		warning	Avoid underscores
25	R.1		warning	Limit the use of implementation-dependent types.
26	R.1		warning	Limit the use of implementation-dependent types.
27	R.1		warning	Limit the use of implementation-dependent types.
Line	Id	CWE	Severity	Message
Function_Block.c				
24	R.1		warning	Limit the use of implementation-dependent types.
24	M.29		warning	Avoid underscores
26	M.16		warning	Avoid use of the ?: operator
28	R.1		warning	Limit the use of implementation-dependent types.
28	M.29		warning	Avoid underscores
32	R.1		warning	Limit the use of implementation-dependent types.
Line	Id	CWE	Severity	Message
Component_FBD.c				
23	M.29		warning	Avoid underscores
29	R.1		warning	Limit the use of implementation-dependent types.
29	M.29		warning	Avoid underscores
43	R.1		warning	Limit the use of implementation-dependent types.
43	M.29		warning	Avoid underscores
44	R.1		warning	Limit the use of implementation-dependent types.
44	M.29		warning	Avoid underscores
45	R.1		warning	Limit the use of implementation-dependent types.
45	M.29		warning	Avoid underscores
Line	Id	CWE	Severity	Message
System_FBD.c				
24	M.29		warning	Avoid underscores
30	R.1		warning	Limit the use of implementation-dependent types.
30	M.29		warning	Avoid underscores
39	M.29		warning	Avoid underscores
40	M.29		warning	Avoid underscores
41	M.29		warning	Avoid underscores

그림 1 FFT의 Cppcheck html report view

```
M.29 24  22  typedef struct{ <--- Avoid underscores
R.1 25   23      int PTSP; <--- Limit the use of implementation-dependent types.
R.1 26   24      int TSP; <--- Limit the use of implementation-dependent types.
R.1 27   25      int TRIP_CNT; <--- Limit the use of implementation-dependent types. <--- Avoid underscores
M.29 27  26      int PTRIP_CNT; <--- Limit the use of implementation-dependent types. <--- Avoid underscores
R.1 28   27      bool TRIP_LOGIC; <--- Limit the use of implementation-dependent types. <--- Avoid underscores
M.29 28  28      bool PTRIP_LOGIC; <--- Limit the use of implementation-dependent types. <--- Avoid underscores
R.1 29   29      bool TRIP; <--- Limit the use of implementation-dependent types.
M.29 29  30      bool PTRIP; <--- Limit the use of implementation-dependent types.
R.1 30   31
```

그림 2 FFT System 파일 Cppcheck html report view의 소스코드 화면

Case Study

표3 FFT, FRT, MFT, VFT, VRT 프로그램의 정적분석 결과

NUREG/CR-6463 가이드라인	규칙	규칙을 위반한 프로그램	
		소스 파일	헤더 파일
4.1.26_1 Limit the use of implementation-dependent types.	R-15	FFT, FRT, MFT,VFT, VRT	FFT, FRT, MFT,VFT, VRT
4.4.1.6_1 Avoid use of the ?: operator	M-16	FFT, FRT, MFT,VFT, VRT	
4.4.1.8_1 #defiine should be place in parentheses, even for a single number.	M-20		FFT, FRT, MFT,VFT, VRT
4.4.5.1 Minimizing Platform-Dependent Data Types	M-28	FFT, FRT, MFT,VFT, VRT	FFT, FRT, MFT,VFT, VRT
4.4.5.2_1 Avoid underscores	M-29	FFT, FRT, MFT,VFT, VRT	FFT, FRT, MFT,VFT, VRT

소스파일에서는 컴파일러와 하드웨어 의존적인 타입 int, char, float을 검출하는 4.1.2.6_1 가이드라인과 4.4.5.1 가이드라인과 삼항연산자를 사용 금지하는 4.4.1.6 가이드라인, 언더스코어(_)의 사용을 금지하는 4.4.5.2_1 가이드라인이 검출되었으며 헤더 파일에서는 #define문을 작성시 소괄호로 값을 구분되어야 하는 4.4.5.2_1 가이드라인, 4.1.2.6_1과 4.4.5.1 가이드라인과 4.4.5.2_1 가이드라인이 위배되었다.

Conclusion

Conclusion

결론 및 향후 연구

개발된 규칙을 사용하여 C로 작성된 원자력 발전소 소프트웨어 프로그램의 NUREG/CR-6463 위반 여부를 정적분석으로 검출할 수 있게 되었다. 이로 인하여 원자력 발전소 소프트웨어 프로그램 코드 리뷰에 드는 소요시간이 상당부분 단축될 수 있을 것으로 예상한다.

향후 NUREG/CR-6463의 가이드라인 중 규칙 집합으로 변환되지 않은 가이드라인을 검증하는 방법에 대해 연구할 계획이다. 또한 FBDtoC 개선의 Testing tool로 사용할 계획도 가지고 있다.