NUREG/CR-6463 가이드라인에 대한 Cppcheck 정적 분석 규칙 집합 개발

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

Developing Cppcheck rule set on NUREG/CR-6463 guideline

SeungHyeon Song○ Sejin Jung Junbeom Yoo
Division of Computer Science and Engineering, Konkuk University

요 약

원자력 발전소와 같은 안전성이 중요한 시설의 소프트웨어는 안전성을 최우선으로 고려하여 개발되어야 한다. 본 논문에서는 원자력 발전소 소프트웨어 프로그램을 위한 코드 리뷰 가이드라인인 NUREG/CR-6463을 기반으로 하여 Cppcheck에서 활용가능한 C프로그램용 규칙집합을 개발하였다. 개발된 규칙집합은 C코드로 작성된 프로그램의 NUREG/CR-6463 위반 사항을 정적분석도구로 분석 가능하게 하였다.

1. 서 론

안전성은 원자력 발전소와 같은 실시간 임베디드 시스템에서 중요한 요소이다.[1] NRC(U.S. Nuclear Regulatory Commission)와 같은 규제 기관에서는 NURG-0800[2]을 통해 SRP(Standard Review Plan)를 제공하여 소프트웨어 개발주기 동안 안전성 검사를 실행하도록 권장하고 있다. NUREG-0800 BTP 7-14 파트에서는 코드 리뷰를 통해 원자력 발전소에 사용될 소프트웨어의 안전성을 확인하도록 하고 그 지침으로서 NUREG/CR-6463[3]을 사용할 것을 제안하고 있다.

NUREG/CR-6463은 코드 리뷰 가이드라인으로서 코드 리뷰 시 체크리스트화 하거나 코딩 컨벤션 지침으로 사용되었다.[4] 이러한 코드 리뷰를 통한 분석은 원자력 발전소와 같은 규모가 크고 복잡성이 높은 시스템의 소프트웨어 검토에 효율적이지 않다. 따라서 NUREG/CR-6463을 정적분석 가능한 형태로 개발하여 도구를 통한 분석이 가능하도록 할 필요가 있다.

본 논문에서는 원자력 발전소에 사용되는 C코드로 작성된 프로그램에 대하여 NUREG/CR-6463 가이드라인 위반사항을 효율적으로 분석하기 위해 Cppcheckeck[5]에서 사용가능한 NUREG/CR-6463에 관한 규칙집합을 개발하였다. C로 작성된 원자력 발전소 소프트웨어 프로그램에 대하여 정적 분석 도구를 통한 NUREG/CR-6463 위반사항을 검출 가능하게 하였다. 개발된 규칙집합이 위반사항을 검출하는지 확인하기 위하여 RPS(Reactor Protection System)에 사용되는 로직을 구현한 C 프로그램을 Cppcheck로 정적분석 하였다.

2. NUREG/CR-6463

NUREG/CR-6463은 NRC(Nuclear Regulatory Commission)에서 제안한 원자력 발전소의 소프트웨어 개발에 사용되는 코드 리뷰 가이드라인으로서 소스코드 리뷰에 사용된다. NUREG/CR-6463은 Ada, C and C++, PLC Ladder Logic, Sequential Function Charts, Pascal, PL /M과 같은 원자력 발전소 소프트웨어 개발에 사용되는 High-Level Language의 코드 리뷰 가이드라인을 제시하고 있다. 프로그래밍 언어 별 섹션은 Reliability, Robustness, Traceability, Maintainability라는 4가지 소프트웨어 안전성 속성을 중심으로 작성 되어있다.

Reliability는 소프트웨어가 성공적으로 실행될 가능성과 요구되는 컨디션 동안 성공적으로 작동할 가능성에 관여하는 속성이다. C언어 파트에서는 메모리 관리, 제어흐름, 작동 타이밍 등이 있다. Robustness는 소프트웨어가 비정상적인 조건이나 이벤트에서 허용가능한 방식으로 작동할 가능성에 관한 속성이다. 이 단락은 특히 소프트웨어 안전성과 밀접하게 연관되어 있으며, C언어는 예외처리, 입출력 확인과 같은 이슈가 있다. Traceability는 소스 코드 및 라이브러리 구성요소 출처 및 개발 프로세스를 검토하고 식별하는 것과 관련이 있는 속성이다. C언어에서는 코드의 가독성, 빌트인 함수의 최소화, 컴파일 된 라이브러리의 최소화, 버전컨트롤, 주석과 문서화 항목이 있다. Maintainability는 소프트웨어의 유지보수에 관한 항목으로 C언어에서는 데이터 추상화와 Malleability, Functional cohesiveness 항목이 있다. 이러한 속성에 따른 규칙들은 원자력 발전소 소프트웨어 프로그램의 programming & auditing 가이드라인으로 활용할 수 있다.

3. NUREG/CR-6463기반의 Cppcheck ruleset 개발

NUREG/CR-6463은 자연어로 작성된 가이드라인이기 때문에 Cppcheck에서 사용하기 위해서는 Cppcheck Custom Rule을 개발해야 한다. Cppcheck Custom Rule은 Pattern과 Message로 이루어져 있다. Pattern은 소스코드에서 검출할 위반사항과

일치하는 문자열을 POSIX 정규표현식[6]으로 나타낸 것이며, Message는 Pattern과 일치하는 소스코드가 발견될 경우 사용자에게 보여지는 정보이다. Cppcheck에서 사용가능한 규칙집합을 개발하기 위해서는 Message에 사용될 정보를 규정하고 NUREG/CR-6463의 가이드라인에서 Pattern에 사용할 토큰정보를 추출하여야 한다. NUREG/CR-6463의 가이드라인은 크게 샘플 코드가 제시되는 경우와 그렇지 않은 경우가 있다. 본 논문에서는 가이드라인의 작성 방식에 따라 샘플코드가 제공되는 경우 샘플 코드를 Cppcheck를 통해 토큰정보를 추출하였으며, 샘플코드가 제공되지 않는 경우 가이드라인에서 명시하고 있는 변수 타입, 함수 이름, 스타일을 토큰정보로 추출하였다. 추출된 토큰을 바탕으로 토큰과 일치하는 소스코드를 검출하는 POSIX 정규표현식을 Pattern으로 작성하였다. 또한 Message의 정보인 id는 각 섹션에서 순서에 따라 번호를 부여하였으며, Severity는 NUREG/CR-6463을 기준으로는 위험도를 구분하기 어렵기 때문에 같은 레벨로 통일하였다. Summary 정보는 NUREG/CR-6463 가이드라인의 핵심문장을 수정하지 않고 사용하였다.

표1은 NUREG/CR-6463의 원문을 규칙으로 개발한 결과의 예시이다. 표1의 4.1.2.2는 NUREG/CR-6463에 코드 샘플이 존재하여 코드 샘플로부터 토큰을 추출하여 Pattern (catch all else if token))을 제작하였고, 4.4.1.6의 경우 코드 샘플은 없으나 원문에서 삼항연산자(?:)의 사용을 금지하는 지침이 명확하게 기술되어 Pattern(catch all ?: token)을 작성 할 수 있었다.

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-6463의 가이드라인 중에서 Cppcheck의 기본 기능과 중복되는 경우 cppcheck의 기능으로 정적분석 하도록 하였다. 예를 들어 “4.1.1.9 Proper Array Indexing”는 Array를 사용할 경우 Array의 크기를 넘는 범위를 주의하여 사용할 것을 제안하고 있다. Cppcheck는 Proper Array indexing(배열의 크기가 올바르게 사용되었는지) 검증하는 기능을 포함하고 있으므로 이러한 경우 정규표현식 개발이 아닌 Cppcheck의 기본 기능으로 검사하도록 Cppcheck의 기능과 규칙을 1:1 매칭하였다.

NUREG/CR-6463의 가이드라인 개수와 변환된 규칙의 개수는 표2와 같다. 표2는 4.1.1 Predictability of Memory Utilization과 같은 속성별 하위 섹션으로 구분하여 각 속성 하위 섹션 별로 NUREG/CR-6463의 가이드라인을 몇 개의 규칙으로 개발하였는지 표기하였다. 표1의 Total에서 NUREG/CR-6463의 가이드라인 개수는 104개 인 것에 비하여 개발된 규칙은 75개인 것을 알 수 있다. 이는 NUREG/CR-6463 가이드라인 중 일부는 정적분석이 불가능 하기 때문이다.

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

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**

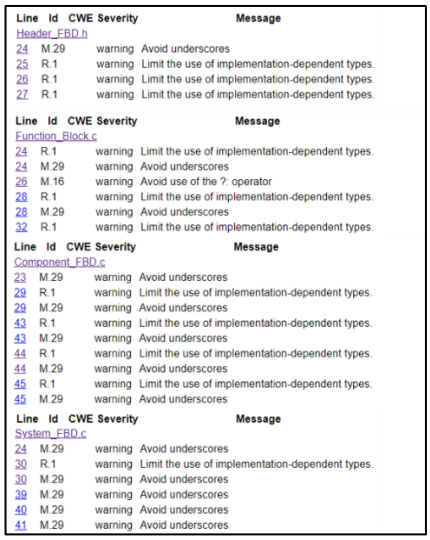
NUREG/CR-6463 가이드라인 중 일부는 정적분석이 불가능하다. 예를 들어 4.3.2 Document all cases of dynamic binding to externally developed libraries 가이드라인은 문서작성에 관한 가이드라인으로 정적분석도구로 확인하는 것이 불가능하다. 이러한 가이드라인들은 개발에서 제외되었기 때문에 전체 104개의 NUREG/CR-6463의 가이드라인 중 29개의 가이드라인을 제외하고 75개의 가이드라인이 정적분석 가능한 규칙 집합으로 개발되었다.

4. Case Study

본 논문에서 개발한 규칙집합이 C프로그램으로부터 NUREG/CR-6463 위반사항을 검출 가능한지 확인하기 위하여 RPS에서 사용되는 FBD(Function Block Diagram)[7]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[9] 기능을 사용하여 C프로그램으로 변환하여 정적분석 하였다. 각 로직은 동일하게 System, Component, Function Block, Header로 구성되어 있다.

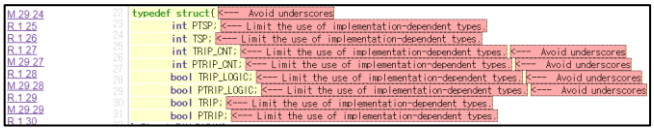
검사환경은 Ubuntu LTS 18.04 버전에서 Cppcheck 1.82 버전으로 개발된 규칙을 사용하여 진행하였다. 그림 1은 FRT의 System.c 파일을 분석한 결과이다. FRT 로직의 System파일에서 4.4.5.2 “Avoid underscores.” 규칙을 11개 라인에서 위반한 것을 확인할 수 있다. 그림 1은 Cppcheck html report 기능을 사용하여 도출된 검사 결과의 일부이다. 위반사항이 검

출된 파일과 Rule 별로 검출된 라인을 확인할 수 있다. 각 라인을 클릭하여 그림2와 같이 소스코드와 위반 사항을 같이 확인할 수 있었다.



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   typedef struct{ <---  Avoid underscores
R.1.25          int PTSP; <--- Limit the use of implementation-dependent types.
R.1.26          int TSP; <--- Limit the use of implementation-dependent types.
R.1.27          int TRIP_CNT; <--- Limit the use of implementation-dependent types. <---  Avoid underscores
M.29.27         int PTRIP_CNT; <--- Limit the use of implementation-dependent types. <---  Avoid underscores
R.1.28          bool TRIP_LOGIC; <--- Limit the use of implementation-dependent types. <---  Avoid underscores
M.29.28         bool PTRIP_LOGIC; <--- Limit the use of implementation-dependent types. <---  Avoid underscores
R.1.29          bool TRIP; <--- Limit the use of implementation-dependent types.
M.29.29         bool PTRIP; <--- Limit the use of implementation-dependent types.
R.1.30
```

그림 2 FFT System 파일 Cppcheck html report view의 소스 코드 화면

다섯가지 로직을 검사한 결과 표 3과 같이 위반사항이 검출되었다. 표3의 소스파일 위반사항에서는 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. 검출된 위반사항의 소스코드를 확인한 결과 false negative로 검출된 위반사항은 없었다.

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

5. 결론 및 향후 연구

본 논문에서는 NUREG/CR-6463위반사항을 정적분석 가능한 규칙을 개발하였다. 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.

Acknowledgement

"본 연구는 2021년도 과학기술정보통신부 및 정보통신기획평가원의 SW중심대학지원사업의 결과로 수행되었음 "(No.2018-0-00213, SW중심대학(건국대학교))

참고문헌

[1] N. G. Leveson, SAFEWARE, System safety and Computers, Addison Wesley, (1995)

[2] USNRC, NUREG-0800, Standard Review Plan for the Review of Safety Analysis Reports for Nuclear Power Plants: LWR Edition, June 1987,

[3] USNRC, Review Guidelines on Software Languages for Use in Safety Systems of Nuclear Power Plants, NUREG/CR-6463 June 1996

[4] 한국원자력연구원, 원전 안전 소프트웨어 위해도 및 안전보증 평가기술 개발 최종보고서, October 20.2017

[5] Daniel marjamaki, http://cppcheck.sourceforge.net,May .10 .2021

[6] The Open Group, IEEE Std 1003.1 2017 (Revision of IEEE Std 1003.1-2008), The Open Group Base Specifications Issue 7, 2018 edition

[7] IEC, IEC 61131-3 "International Standard" Part 3: Programming languages. 1993.3

[8] Dong-Ah Lee, Eui-Sub Kim, Junbeom Yoo, Quantitative measures of thoroughness of FBD simulations for PLCbased digital I&C system, Nuclear Engineering and Technology Volume 53, Issue 1, January 2021, Pages 131-141

[9] Eui-Sub Kim, Dong-Ah Lee, Sejin Jung, Junbeom Yoo, Jong-Gyun Choi, Jang-Soo Lee "NuDE 2.0: A Formal Method-based Software Development, Verification and Safety Analysis Environment for Digital I&Cs in NPPs," Journal of Computing Science and Engineering, Vol.11, No.1, pp.9-23, 2017.