JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 30 No. 8, pp. 11-19, August 2025
https://doi.org/10.9708/jksci.2025.30.08.011

A Comparative Study on the Performance of Commercial and Open-source Static Analysis Tools for MISRA C Coding Standards Compliance

Donghee Ha*

*Researcher, Agency for Defense Development, Daejeon, Korea

[Abstract]

In safety-focused industries such as defense and automotive, MISRA C is widely adopted to ensure reliable software. However, static analysis tools differ in rule coverage and detection performance, and comparative studies remain limited. In this paper, we evaluate a commercial software tool A and an open-source software CPPCHECK using the MISRA C:2012 example suites. We analyze the static analysis results produced by these tools across three categories (Test Case, Rule Category, and Rule Section) using four metrics (accuracy, precision, recall, and F1-score). Experimental results show that Tool A performs better than CPPCHECK in two categories (Test Case and Rule Category). However, in Rule Sections 9, 15, and 16, CPPCHECK achieves better performance, indicating strengths in specific areas.

▸ **Key words**: Static Analysis, MISRA C Coding Rule, Static Analysis Tool, Commercial Software, Open-source Software

[요 약]

국방, 자동차와 같이 안전성이 중요한 산업에서는 신뢰성 있는 소프트웨어 개발을 보장하기 위해 MISRA C가 널리 적용되고 있다. 그러나 정적분석 도구는 규칙 지원 범위와 탐지 성능에서 차이가 있으며, 이에 대한 비교 연구는 제한적이다. 본 연구에서는 MISRA C:2012 예제 모음을 이용하여 상용 소프트웨어 도구 A와 오픈소스 소프트웨어 CPPCHECK를 평가하였다. 두 도구가 수행한 정적 분석 결과를 테스트 케이스(Test Case), 규칙 범주(Rule Category), 규칙 섹션(Rule Section)의 세 가지 범주에서 분석하였으며, 정확도(accuracy), 정밀도(precision), 재현율(recall), F1 점수(F1-score)의 네 가지 평가지표를 사용하였다. 실험 결과, 도구 A는 두 가지 범주(Test Case, Rule Category)에서 CPPCHECK보다 우수한 성능을 보였다. 그러나 규칙 섹션 9, 15, 16에서는 CPPCHECK가 더 좋은 성능을 보였다.

▸ **주제어**: 정적 분석, MISRA C 코딩규칙, 정적 분석 도구, 상용 소프트웨어, 오픈소스 소프트웨어

• First Author: Donghee Ha, Corresponding Author: Donghee Ha
*Donghee Ha (dhha@add.re.kr), Agency for Defense Development
• Received: 2025. 07. 28, Revised: 2025. 08. 08, Accepted: 2025. 08. 18.

Copyright © 2025 The Korea Society of Computer and Information
http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945
www.kci.go.kr

I. Introduction

최근 국방, 자동차, 항공, 철도 등의 산업에서 소프트웨어에 대한 요구사항이 증가하고 기능이 다양해지고 있다. 소프트웨어의 활용도가 증가됨에 따라 검증과 품질에 대한 중요도도 증가하고 있다. 소프트웨어의 검증과 품질 향상을 하기 위해 정적분석을 수행한다. 정적분석은 소프트웨어를 실행시키지 않은 상태에서 소스코드만 보고 분석하여 발생할 수 있는 오류나 보안 약점 등을 찾는 과정을 의미한다. 정적 분석을 통해 잠재적인 결함을 사전에 방지할 수 있고, 안정성 측면의 품질을 높일 수 있다. 추후 시험 및 관리 비용의 절감 효과도 있다.

정적분석을 수행시에는 산업의 특징에 따라 준수하며 지켜야 하는 규칙들이 존재한다. C/C++ 언어의 대표적인 정적분석 규칙은 MISRA(Motor Industry Software Reliability Association) C/C++[1]와 CWE(Common Weakness Enumeration)-658/659[2] 등이 있다. 먼저 MISRA C/C++는 MISRA에서 개발한 코딩규칙으로 임베디드 시스템에서 C/C++의 코드에서 발생할 수 있는 오류에 관한 코딩규칙이다. 국방, 자동차 등의 산업에서는 MISRA C/C++ 코딩규칙을 지키도록 규정하고 있다[3]. CWE-658/659는 MITRE 비영리 단체에서 만든 보안 약점 목록으로 C/C++ 언어로 개발된 소프트웨어 및 하드웨어의 보안 약점에 관한 규칙이다. 보안이 중요한 국방, 항공 등의 산업에서 CWE 보안약점을 준수하여 소프트웨어가 보안에 강하도록 개발하고 있다. 이러한 코딩규칙이나 보안 약점들의 규칙 준수를 위한 정적분석은 자동화 도구인 소프트웨어 검증 도구를 이용하여 수행되고 있다.

정적분석을 위한 소프트웨어 검증 도구는 상용 소프트웨어 뿐만 아니라 공개 소프트웨어에서도 개발되어 왔다. 다양한 도구들이 존재하는 만큼, 도구마다 지원하는 규칙도 다르며, 규칙에 따라 탐지 성능도 다르다는 이슈가 있고, 효과적인 정적분석 도구를 선택하는 것이 어렵다[4]. 이에 따라 각 도구의 공정한 비교가 필요하여 각 검증 도구의 성능을 비교 분석하는 연구가 수행되어 왔다. CWE 보안 약점은 RATS[5], Flawfinder[6], CPPCHECK[7] 등의 공개 소프트웨어 검증도구가 활발히 개발되었다. 이러한 검증도구를 활용하여 CWE 보안 약점을 정적분석하여 성능을 비교 분석하는 연구도 수행되었다[8,9]. 그에 비해 MISRA C/C++ 코딩 규칙에 관한 검증 도구들의 성능 비교 연구는 부족하다. 검증 도구에서 MSIRA C/C++를 지원하기 위해서는 라이선스가 필요하며 사용자가 별도로 MISRA C/C++ 문서를 확인하려면 구매하여야 한다. MISRA C/C++를 지원하는 검증도구들은 상용 소프트웨어가 주를 이루고 있으며 공개 소프트웨어 검증도구 중 많이 사용되고 있는 CPPCHECK만 MISRA C의 라이선스를 구매하여 지원하고 있다. 공개 소프트웨어 검증도구가 부족하다보니 MISRA C/C++의 검증 도구들의 성능을 비교 분석하는 연구가 부족하다.

본 논문에서는 상용 소프트웨어와 공개 소프트웨어 정적분석 도구의 비교 분석 연구를 수행했다. 정적분석 수행하기 위해서 MISRA C 코딩규칙의 예제 모음(Example Suite)[10,11,12]을 이용하였다. MISRA C 예제 모음은 MISRA 규칙에 대한 예제를 작성한 코드로 각 규칙별로 구현되어 있다. MISRA C를 지원하는 공개 소프트웨어 검증 도구인 CPPCHECK와 상용 소프트웨어 정적분석 도구 A를 활용하여 MISRA C 예제 모음의 정적분석을 수행하였다. 2가지 도구의 정적분석 수행 결과를 토대로 혼동배열(Confusion Matrix)을 정리하여 3가지 기준(검증 도구별 테스트 케이스 기준, 규칙 카테고리(Rule Cateogry) 별 테스트 케이스 기준, 규칙 섹션(Rule Section) 별 테스트 케이스 기준)으로 분석하였다. 평가지표로는 정확도(Accuracy), 정밀도(Precision), 재현율(Recall), F1 점수(F1 Score)로 4가지 지표를 이용하여 도구의 성능을 비교하였다. 공개 소프트웨어와 상용 소프트웨어 검증도구에서 정적 분석을 수행한 결과는 상용 도구인 A가 전체적으로 평가지표에서 좋은 성능을 보였지만, 규칙 섹션 중 9, 15, 16에서는 CPPCHECK가 더 좋은 결과를 보였다. 또한, 5개의 규칙에 대해서는 2가지 도구 모두 탐지하지 못함을 확인하였다.

2장에서는 관련 연구을 설명하고, 3장에서는 실험 내용 및 평가 방법에 대해 설명한다. 4장에서 실험 결과를 설명하고, 5장 결론으로 마무리한다.

II. Related Works

정적분석은 소프트웨어를 실행하지 않고, 소스코드만을 가지고 분석하는 방법이다. 정적분석을 수행함에 있어 지켜야 할 다양한 규칙들이 존재한다. C/C++ 언어와 관련된 보안 약점으로는 CWE-658/659가 있으며, 코딩규칙으로는 MISRA C/C++가 있다. CWE는 MITRE 비영리 단체에서 만든 목록으로 소프트웨어 및 하드웨어 보안 약점 유형들의 목록들이 작성되어 있다. CWE 보안 약점을 평가하기 위해 줄리엣 테스트 모음이 개발되었다[13]. 줄리엣 테스트 모음은 미국 국가보안국의 CAS(Center for

www.kci.go.kr

Assured Software)에서 개발되었으며, Table 1과 같이 다양한 정적분석 도구에서 성능 비교 연구에서 사용되었다[8,9,14,15]. [14,15]에서는 Java에 대해 공개/상용 소프트웨어 검증 도구를 사용하여 정적분석 결과를 비교분석하였다. [14]에서는 총 8개의 공개/상용 정적분석 도구를 사용하였으며, 20개의 CWE 항목을 실험하였다. 실험 결과, Java Pathfinder[16], YASCA[17], Bandera[18] 순으로 가장 높은 정밀도를 보였다. [15]에서는 Infer[19], SonarQube[20], SpotBugs[21], Find Security Bug[22], PMD[23]로 총 5개의 공개 소프트웨어 검증도구를 사용하여 CWE 항목 70개를 분석하였다. Infer에서 오탐이 가장 낮았으며, PMD는 F1 점수가 가장 높았다. [8,9]에서는 Java 뿐만 아니라 C/C++의 CWE 항목에 대해 5개의 공개 소프트웨어 검증도구로 정적분석하여 결과를 비교하였다. C/C++에서는 3개의 공개 소프트웨어 검증도구(flawfinder[5], RATS[6], CPPCHECK[7])를 사용하였고, Java에서는 2개의 공개 소프트웨어 검증 도구(SpotBugs[21], PMD[23])를 사용하였다. 실험결과로 C/C++에서는 CPPCHECK가 flawfinder, RATS보다 많은 오탐을 확인하였으며, Java에서는 Spot bugs이 PMD보다 많은 취약점을 탐지함을 확인하였다.

MISRA C/C++는 MISRA에서 C/C++ 언어의 개발 표준으로 임베디드 시스템에서의 안정성, 호환성, 신뢰성을 목표로 한다. Table 2는 MISRA 코딩 규칙을 적용하여 수행한 연구를 나타낸다. 공개 소프트웨어 중 많이 사용되는 C++로 개발된 PX4 공개 소프트웨어[24]에 MISRA 코딩규칙을 적용하여 정적분석을 수행하여 분석한 연구가 수행하였다[25]. 해당 연구에서는 C++의 코딩규칙인 MISRA C++:2008을 적용하여 정적 분석을 수행하였다. 분석 결과 54% 정도의 코딩 규칙을 위배하고 있어 모듈 단순화 등의 개선 방안을 제안하고 있다. 정적분석 결과인 알람에 대해 전체적으로 분석을 하였지만, 도구로부터 나온 알람이 정탐인지, 오탐인지에 대한 분석은 이루어지지 않았다. MSIRA C 관련한 연구로는 안정성이 요구되는 시스템에 MISRA C 코딩규칙을 적용하여 정적분석을 수행하여 오류의 빈번도 기준으로 분석하였다[26,27]. 하지만, 이전 버전인 MISRA C:2004를 적용하였으며, 알람의 올바른 알람인지 확인 할 수 있는 정탐, 오탐 여부는 분석되지 않았다. [28]에서는 MISRA C:2012를 적용하여 RTOS들의 소스코드를 CPPCHECK 도구로 정적분석하여 비교 분석하였다. 대부분 RTOS에서는 Mandatory 규칙에서는 비교적 잘 준수하지만, Required, Advisory 규칙에서는 위반이 많은 걸 볼 수 있었다. 하지만, CPPCHECK 도구만을 사용하여 CPPCHECK의 성능에 의존되어 있다.

본 연구에서는 C언어의 코딩 규칙을 구현한 MISRA C:2012 예제 모음을 이용하여 정적분석을 수행하였다. 수행 결과인 알람들을 정탐, 오탐, 미탐으로 구분하여 도구의 성능 비교 연구를 수행하였다.

Table 1. Summary of related work about CWE

Ref	Programming Language	Free/ Commercial	#Tools	#Vul
[14]	Java	Free	8	20
[15]	Java	Both	5	70
[8,9]	Java, C/C++	Free	5	15

Table 2. Summary of related work about MISRA C/C++

Ref	Targeted Application	Version of MISRA
[25]	PX4	MISRA C++:2008
[26,27]	Embedded System	MISRA C:2004
[28]	RTOS	MISRA C:2012

III. Experimental Setup

1. Static Analysis Tools

본 논문에서는 공개 소프트웨어 검증 도구인 CPPCEHCK와 상용 소프트웨어 A 도구를 사용하였다.

CPPCHECK는 C, C++로 개발된 코드의 정적분석을 수행하는 정적분석 도구이다. 다양한 개발 환경에서 플러그인 형태로 사용이 가능하고, 윈도우, 리눅스 등의 다양한 환경에서 사용할 수 있다. 정의되지 않은 행동(Undefined behavior), 위험한 코드 패턴, 코딩 스타일 등을 분석할 수 있다. 또한, CWE-658/659, MISRA C/C++와 같은 규칙도 지원한다. MISRA 규칙은 라이선스를 구매해 지원하지만, MISRA를 설정하기 위해서는 MISRA 문서를 유료로 구매하여 문서를 토대로 규칙을 정리해야 한다.

A 도구는 C, C++를 지원하며, MISRA C/C++, CWE-658/659, CERT C 등의 정적분석 규칙을 지원하는 검증 도구이다. 코드, 임베디드 프로젝트, 빌드 명령어 프로젝트 등을 지원한다. 여러 컴파일러에서 분석이 가능하며, 비주얼 스튜디오와 같은 통합개발환경 프로젝트도 지원한다.

www.kci.go.kr

Table 3. The number of Rule and Test case on Rule Section

Rule Section	Number of Rules	Number of Test Cases	
		Non-compliant	Compliant
1	-	-	-
2	7	15	8
3	2	5	1
4	2	3	3
5	9	17	9
6	2	3	2
7	4	12	15
8	14	45	27
9	5	15	20
10	8	52	32
11	9	23	14
12	5	12	16
13	6	25	18
14	4	14	11
15	7	15	7
16	7	11	6
17	8	17	7
18	8	30	41
19	2	3	1
20	14	23	18
21	20	50	13
22	10	15	9
Total	153	405	278

Table 4. The number of total Rule, Non-complaint and Compliant

Total Number of Rules	Number of Test Cases	
	Non-compliant	Compliant
153	405	278

Table 5. The number of Rule and Test case on Rule Category

Rule Category	Number of Rules	Number of Test Cases	
		Non-compliant	Compliant
Mandatory	16	31	17
Required	106	297	187
Advisory	31	77	74
Total	153	405	278

MISRA C:2012의 Directive 규칙은 A 도구는 지원하지만, CPPCHECK는 지원하지 않아 Directive 규칙은 분석에서 제외하였다. CPPCHECK는 MISRA C:2012의 규칙 1.1과 1.2는 지원하지 않으며, 규칙 12.5가 누락되어 153개의 규칙을 지원한다. A 도구는 156개의 규칙을 전부 지원한다.

2. MISRA C:2012 Example suite

MISRA C:2012 예제 모음[10,11,12]은 각각의 규칙을 코드로 구현한 예제들을 모아두었다. MISRA C:2012 예제 모음은 C파일과 헤더파일을 포함하여 총 299개의 파일로 구성되어 있다. MISRA C:2012의 각 규칙은 1개이상의 C 파일로 구현되어 있고, 필요시 헤더(Header) 파일도 포함되어 있다. 또한, 각각의 예제 파일에는 하나의 규칙만으로 작성되어 있다. 파일에 작성된 코드 라인은 하나의 테스트 케이스이며, 테스트 케이스는 Compliant와 Non-compliant로 구성되어 있다. Compliant는 규칙에 걸리지 않는 안전한 코드 라인이며 도구에서 경고 알람이 나오지 않아야 한다. 반면, Non-compliant는 규칙에 합당하지 않는 오류를 발생시킬 수 있는 코드 라인으로 경고 알람이 나와야 한다. MISRA C:2012는 156개의 규칙을 총 22개의 섹션으로 구분되어 있다. 각 섹션은 비슷한 문법이나 항목을 가지고 있는 여러개의 규칙을 하나로 묶어놓았다. Table 3은 섹션별로 규칙과 테스트 케이스의 개수를 작성한 표이다. 규칙 섹션 1에 대해서는 예제 모음에 예제 코드가 구현되어 있지 않아 테스트 케이스가 없다. 규칙 섹션 1에 포함된 규칙을 제외한 모든 규칙은 1가지 이상의 Non-compliant로 작성되어 있으며, Compliant는 규칙에 따라 존재하지 않는 경우도 있다. Table 4와 같이 MISRA C:2012 예제 모음은 규칙 1 섹션을 제외한 총 153개 규칙이 구현되어 있으며, 총 683개의 테스트 케이스로 이루어져 있다. 테스트 케이스는 각 405개의 Non-compliant와 278개의 Compliant로 구성되어 있다.

MISRA C:2012는 심각한 오류를 야기할 수 있는 위험도에 따라 3가지 카테고리로 규칙을 분류했다. Mandatory가 가장 위험한 오류를 발생할 수 있는 규칙을 나타내고, Advisory가 가장 덜 위험한 문제를 발생시킬 수 있는 규칙을 나타낸다. Table 5는 카테고리에 따른 규칙과 테스트 케이스의 개수를 나타낸 표이다.

본 논문에서는 정적분석 결과를 683개 테스트 케이스 기준, 22개 규칙 섹션 기준, 3가지 규칙 카테고리 기준으로 분석하였다.

3. Evaluation Metrics

CPPCHECK와 A 도구에서 MISRA C:2012 예제 모음에 대해 정적분석을 실행한 결과를 비교 분석하였다.

도구의 탐지 결과를 분석하기 위해 혼동 배열(Confusion Matrix)을 이용하였다. 혼동 배열은 도구의 탐지 결과를 시각화하도록 작성되며 정적 분석 도구의 성능 평가를 위해 대표적으로 사용되고 있다[29,30].

www.kci.go.kr

Table 6. Concept of confusion matrix with static analysis and validation tools

		Tool Alarm	
		Positive (Alarm)	Negative (Non-alarm)
Source Code	Positive (Non-compliant)	True Positive	False Negative
	Negative (Compliant)	False Positive	True Negative

Table 7. Confusion matrix of result of static analysis on the CPPCHECK

	CPPCHECK	Tool Alarm	
		Positive (Alarm)	Negative (Non-alarm)
Source Code	Positive (Non-compliant)	237 (TP)	168 (FN)
	Negative (Compliant)	13 (FP)	265 (TN)

Table 8. Confusion matrix of result of static analysis on the A tool

	Tool A	Tool Alarm	
		Positive (Alarm)	Negative (Non-alarm)
Source Code	Positive (Non-compliant)	332 (TP)	73 (FN)
	Negative (Compliant)	10 (FP)	268 (TN)

Table 6은 혼동배열에 대한 설명이다. 혼동 배열은 True Positive(TP), False Positive(FP), False Negative(FN), True Negative(TN)으로 이루어진다. TP는 정탐으로 규칙에 위배되는 코드를 제대로 탐지한 알람이다. 본 논문에서는 Non-compliant를 제대로 탐지한 경우를 나타낸다. FP는 오탐을 나타내며 탐지된 내용이 규칙과 전혀 관계없는 알람이다. 본 논문에서는 Compliant를 잘못 탐지하여 나온 알람을 나타낸다. FN는 미탐을 나타내며 규칙에 위배되는 내용이 있어 탐지를 해야하지만 알람이 발생하지 않은 경우를 나타낸다. 즉, Non-compliant의 코드에서 알람이 나와야 하지만, 나오지 않은 경우이다. TN은 규칙에 위배되지 않는 내용을 탐지하지 않은 경우를 나타낸다. 본 논문에서는 Compliant에서 탐지되지 않아 알람이 나오지 않은 경우를 말한다.

정적분석 도구의 성능을 평가하기 위한 평가지표는 정확도(Accuracy, Acc), 정밀도(Precision, Prec), 재현율(Recall, Rec), F1 점수(F1-score, F1)를 이용하였다. 정확도는 얼마나 정확한지를 나타내는 지표이다. 전체 테스트 케이스(TP, FN, FP, TN) 중 정탐(TP, TN)의 비율을 나타낸다. 정밀도는 전체 나온 알람들(TP, FP) 중에서 실제 Non-compliant에 해당하는 정탐(TP)의 비율을 말한다. 정밀도는 도구의 알람의 관점으로 확인할 수 있는 평가지표이다. 재현율은 정탐(TP)과 미탐(FN) 중 정탐(TP)의 비율을 말하며 전체 Non-compliant에 해당하는 코드들(TP, FN) 중에 정탐의 비율과 같다. 정밀도와 달리 재현율은 실제 코드의 관점의 평가 지표이다. F1 점수는 정밀도와 재현율의 조화평균으로 2가지 평가 항목을 균형있게 반영하는 지표이다. 2가지 지표가 함께 높을수록 F1 점수도 높게 나타난다. 4가지 평가 지표의 수식은 아래와 같다. 4가지 지표들은 수치가 높을수록 좋은 성능을 의미한다.

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN} \quad (1)$$

$$\mathrm{Pr}ecision = \frac{TP}{TP+FP} \quad (2)$$

$$Recall = \frac{TP}{TP+FN} \quad (3)$$

$$F1Score = 2 \times \frac{\mathrm{Pr}ecision \times Recall}{\mathrm{Pr}ecision + Recall} = \frac{2TP}{2TP+FP+FN} \quad (4)$$

본 논문에서는 예제 파일의 규칙에 해당하지 않는 알람은 제외하고 해당 규칙의 알람만을 이용하여 평가하였다. 예를 들어 규칙 5.1에 해당하는 내용을 도구에서 정적 분석했을 때, 규칙 5.1뿐만 아니라 3.1에 대한 알람이 나왔지만, 3.1에 대한 알람은 제외하고 5.1에 관한 알람만 이용하였다. 본 논문의 목적은 각 규칙에 대해 도구에 대한 비교 분석이므로 예제 파일에 해당하는 규칙의 알람만을 가지고 분석하였다. CPPCHECK에서 분석한 알람의 개수는 258개이고, A 도구에서 분석한 알람의 개수는 342개이다.

Table 9. The result of analyzing alarms based on the test case

Static Analysis Tool	TP	FP	FN	TN	Acc	Prec	Rec	F1
CPPCEHCK	237	13	168	265	0.735	0.948	0.585	0.724
Tool A	332	10	73	268	**0.878**	**0.971**	**0.820**	**0.800**

www.kci.go.kr

Table 10. The result of analyzing alarms based on the rule.

Static Analysis Tool	Number of Rules	Fully Detected	Partially Detected	Not Detected
CPPCHECK	153	77	42	34
Tool A		113	31	9

IV. Experimental Results

본 논문에서는 MISRA C:2012 예제 모음을 CPPCHECK와 A 도구의 정적 분석한 결과는 총 3가지 기준으로 분석하였다. 테스트 케이스 기준, 규칙 카테고리(Rule Category) 기준, 규칙 섹션(Rule Section) 기준으로 정적 분석 결과를 분석하였다.

1. Analysis based on Test Case

MISRA C:2012 예제 모음은 153개의 규칙을 683개의 테스트 케이스로 작성되어 있다. Non-compliant와 Compliant는 각각 405개, 278개로 구성되어 있다. 683개의 테스트 케이스를 CPPCHECK와 A 도구에서 정적분석을 수행하였다. Table 7과 8은 각 도구에서 수행한 정적분석 결과를 혼동 행렬을 이용하여 시각화한 표이다. CPPCHECK와 A 도구은 Negative 정탐(TN)과 오탐(FP)는 거의 비슷한 수준의 성능을 보였다. 하지만, A 도구가 CPPCHECK보다 95개 테스트 케이스를 탐지(Positive 정탐)하였고, 탐지하지 못한 결함(미탐)이 95개 적었다. Table 9는 Table 7, 8의 혼동행렬을 이용하여 평가지표를 계산하여 나타낸 표이다. 먼저 4가지 평가지표에서 모두 A 도구의 성능이 CPPCHECK보다 좋은 것을 확인 할 수 있다. 정확도와 재현율이 A 도구가 CPPCHECK보다 0.143, 0.235이 높아 다른 평가지표보다 성능 차이가 컸다. 재현율의 경우, CPPCHECK가 0.585로 낮은 성능을 보이고 있다. 재현율은 실제 결함 중에 나온 알람을 의미하는 데, 재현율이 낮다는 것은 정적분석 후에도 탐지하지 못한 결함이 존재 할 수 있음을 의미한다. 정적분석 도구는 최대한 결함을 찾아내야 하므로 재현율의 관점에서는 A 도구를 활용하는 것이 좋다. 2가지 도구의 정밀도와 F1 점수의 차이는 0.023, 0.076으로 비교적 적었다. 정밀도의 경우, 검증 도구에서 나온 알람 중 오탐의 비율을 나타내는 데, A 도구의 평가 지표의 수치가 0.974로 CPPCHECK의 0.948보다 높게 나왔지만, 2가지 도구 모두 0.9가 넘는 높은 수치를 확인하였다. 재현율 기준에서 A 도구가 활용할 말한 성능을 보였고, 정밀도 기준에서는 2가지 도구 모두 활용할 만한 성능을 보여주었다.

실제 코드의 정적분석에서는 치명적인 결함을 발생시킬 수 있는 Non-compliant를 제대로 탐지하는 것이 중요하다. 본 논문에서는 Non-compliant가 제대로 탐지되었는지를 확인하기 위해 153개의 규칙별로 전부 탐지(Fully Detected), 일부 탐지(Partially Detected), 전부 미탐지(Not Detectd)로 분류하여 분석하였다. 전부 탐지는 하나의 규칙에 포함되어 있는 Non-compliant를 전부 탐지한 경우이고, 전부 미탐지는 하나의 Non-compliant도 찾지 못한 경우를 나타낸다. Table 10은 규칙별로 탐지 결과를 분석한 결과이다. 전부 탐지한 규칙의 수가 CPPCEHCK보다 A 도구가 36개 더 많았다. 일부 탐지는 CPPCHECK가 A 도구보다 11개 많았다. CPPCHECK와 A 도구의 전부 미탐지는 각각 34개와 9개로 CPPCHECK가 25개 더 많았다. 또한, 5개의 규칙은 2개 도구 모두에서 탐지하지 못함을 확인하였다. 탐지하지 못한 5개의 규칙 중 Mandatory 규칙은 3개로 규칙 9.1, 21.17, 21.18이며, Required의 규칙은 2개로 규칙 5.2와 5.4이다. 총 5개의 규칙은 2가지 도구에서 모두 탐지하지 못하므로 다른 도구를 사용하거나 매뉴얼 리뷰 등의 방법으로 추가적인 검토가 필요하다.

2. Analysis based on Rule Category

MISRA C:2012는 Mandatory, Required, Advisory로 총 3개의 카테고리로 분류되며 순서대로 치명적인 오류를 발생할 있는 문제의 중요도를 나타낸다. Table 5에서 확인할 수 있듯이 Mandatory 규칙과 테스트 케이스가 가장 적지만, 가장 중요한 규칙이다. Required 규칙은 가장 많고, Advisory가 두 번째로 많다. Table 11은 각 규칙 카테고리 별로 테스트 케이스를 기준으로 CPPCHECK와 A 도구의 정적 분석을 수행한 결과이다.

Mandatory에서는 A 도구가 CPPCHECK보다 좋은 성능을 보이고 있다. 정확도에서는 CPPCHECK가 0.5로 낮은 성능을 보이고 있고, A 도구는 0.750로 비교적 높은 성능을 보이고 있다. 정밀도를 보면 2가지 도구 모두 1로 높은 수치를 보이고 있다. 이는 오탐(FP)이 전혀 나오지 않았기 때문인데, 도구에서 나온 Mandatory 관련 알람은 전부 검토 및 수정이 필요한 알람임을 알 수 있다. A 도구의 재현율이 0.613으로 높았지만, CPPCHECK의 재현율은 0.226으로 낮았다. 재현율이 낮음은 실제 결함들 중에 알람이 나오지 않은 결함이 많다는 것을 알 수 있다. F1 점수는 역시 CPPCHECK는 0.368로 낮지만, A 도구는 0.760으로 CPPCHECK보다 높았다. Required와

www.kci.go.kr

Table 11. The result of analyzing alarms based on the rule category

Rule Category	Static Analysis Tool	TP	FP	FN	TN	Acc	Prec	Rec	F1
Mandatory	CPPCEHCK	7	0	24	17	0.500	1.000	0.226	0.368
	Tool A	19	0	12	17	**0.750**	1.000	**0.613**	**0.760**
Required	CPPCEHCK	173	10	124	177	0.723	0.945	0.582	0.721
	Tool A	244	7	53	180	**0.876**	**0.972**	**0.822**	**0.891**
Advisory	CPPCEHCK	57	3	20	71	0.848	0.950	0.740	0.832
	Tool A	69	3	8	71	**0.927**	**0.958**	**0.896**	**0.926**

Advisory 규칙에 비해 Mandatory 규칙에서 A 도구와 CPPCHCEK의 평가지표를 통해 성능 차이가 큰 것을 확인하였다.

Required 규칙의 평가지표를 보면 A 도구가 CPPCHECK보다 전부 높은 것을 확인할 수 있다. 두 도구의 정확도 차이는 0.153로 A 도구가 높았다. 재현율은 0.240으로 큰 차이를 보이는데, CPPCHECK가 A 도구에 비해 미탐(FN)이 많음을 의미한다. CPPCEHCK은 124개이고, A 도구는 53개로 2배보다 많다. 이에 반해 정밀도의 차이는 0.027로 작았으며, 오탐의 개수는 비슷함을 알 수 있다. F1 점수는 0.17의 차이로 A 도구가 더 높았다. 모든 평가지표가 CPPCHECK보다 A 도구가 높아 Required 규칙에서는 A 도구의 성능이 더 좋음을 확인하였다.

Advisory 규칙도 Required 규칙과 마찬가지로 모든 평가지표에서 A 도구가 CPPCHECK보다 높은 수치를 보이고 있다. 반면 두 도구의 평가지표가 다른 2가지 카테고리보다 비슷한 수치를 보이고 있다. 정확도, 정밀도, 재현율, F1 점수를 보면 각각 0.079, 0.008, 0.156, 0.094의 차이로 비슷한 수치를 보이고 있어 Advisory 규칙에서는 두 도구의 성능이 비슷함을 알 수 있다.

규칙 카테고리 별 테스트 케이스 기준 분석 결과, 전반적인 성능은 A 도구가 CPPCHECK보다 좋은 성능을 보이고 있다. Mandatory, Required, Advisory 순으로 A 도구의 성능 차이가 확연했다. 반면 정밀도를 제외한 정확도, 재현율, F1 점수에서 두 도구 모두 Mandatory의 평가지표가 Required, Advisory의 평가지표보다 낮았다. Mandatory 카테고리의 A 도구의 정확도는 0.750이지만, Required과 Advisory 카테고리의 0.876, 0.927이다. Mandatory는 치명적인 오류를 범할 수 있는 가장 중요한 규칙이지만, 3가지 카테고리에서 평가지표가 가장 낮음을 확인하였다. 두 도구 모두에서 가장 중요한 규칙에 대한 추가적인 검토가 필요하다.

3. Analysis based on Rule Section

MISRA C:2012 예제 모음에서는 153개의 규칙이 총 22개의 섹션으로 나누어져 있다. 각 섹션은 비슷한 문법이나 내용을 가진 규칙들로 묶어서 구분되어 있다. Table 12는 CPPCHECK와 A 도구에서 규칙 섹션별 테스트 케이스를 기준으로 정적 분석 수행 결과이다. 규칙 섹션 1은 테스트케이스가 없으므로 정적분석에서 제외하였다. 정적분석 수행 결과, 대부분의 규칙 섹션에서 A 도구의 평가 지표가 CPPCHECK보다 좋은 성능을 확인할 수 있다. 특히, 14개 규칙 섹션(2, 5, 6, 8, 10, 11, 12, 13, 14, 18, 19, 20, 21, 22)에서 정확도, 재현율, F1 점수의 평가 지표가 CPPCHECK보다 A 도구가 높음을 확인하였다. 규칙 섹션 3, 4, 7, 17의 경우, CPPCHECK와 A 도구가 비슷한 성능을 보였다. 규칙 섹션 9, 15, 16에서는 CPPCHECK가 A 도구보다 좋은 성능을 보인다. 규칙 9는 초기화(Initialization)와 관련규칙, 규칙 15는 제어 흐름(Control flow)이고, 규칙 16은 스위치 문(Switch Statements) 관련 규칙이다. 초기화, 제어 흐름, 스위치 문과 관련하여 주의가 필요한 소프트웨어의 경우, CPPCHECK를 이용하여 정적분석을 수행하여 검증할 수 있다. 특히 규칙 9는 규칙 카테고리 중 가장 중요한 Mandatory가 포함되어 있어 CPPCHECK를 활용하여 심각한 오류 발생을 방지할 수 있다.

규칙 섹션을 기준으로 분석한 결과, 전체적인 규칙 섹션에 대해서는 CPPCECK보다 A 도구를 활용하는 것이 좋다. 하지만, 규칙 9, 15, 16의 주의를 요하는 소프트웨어의 경우 CPPCHECK 도구로 활용하여 정적분석을 수행할 수 있다.

V. Conclusions

본 논문에서는 국방, 철도 등에서 사용되는 MISRA C:2012 규칙의 예제 모음을 활용하여 공개 소프트웨어 검증 도구인 CPPCHECK과 상용 소프트웨어인 A 도구를 활용하여 비교 분석 연구를 수행하였다.

알람 결과는 테스트 케이스 기준, 규칙 카테고리(Rule Category) 기준, 규칙 섹션(Rule Section) 기준으로 총 3가지 기준으로 분석하였다. 첫 번째로 테스트 케이스를 기준으로 분석한 결과, A 도구의 정확도는 0.878이고, CPPCHECK의 정확도는 0.735로 A 도구의 정확도가

www.kci.go.kr

Table 12. The result of analyzing alarms based on the rule section

Static Analysis Tool	CPPCHECK				Tool A			
Evaluation metrics	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1
Rule 1 Section	-	-	-	-	-	-	-	-
Rule 2 Section	0.478	0.800	0.267	0.400	**0.826**	**0.923**	**0.800**	**0.857**
Rule 3 Section	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Rule 4 Section	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Rule 5 Section	0.615	1.000	0.412	0.583	**0.808**	1.000	**0.706**	**0.828**
Rule 6 Section	0.800	1.000	0.667	0.800	**1.000**	1.000	**1.000**	**1.000**
Rule 7 Section	0.963	1.000	0.917	0.957	0.963	1.000	0.917	0.957
Rule 8 Section	0.653	0.857	0.533	0.658	**0.847**	**0.925**	**0.822**	**0.871**
Rule 9 Section	**0.914**	1.000	**0.800**	**0.889**	0.800	1.000	0.533	0.696
Rule 10 Section	0.679	0.931	0.519	0.667	**0.893**	**0.978**	**0.846**	**0.907**
Rule 11 Section	0.838	1.000	0.739	0.850	**0.919**	1.000	**0.870**	**0.930**
Rule 12 Section	0.857	**0.900**	0.750	0.818	**0.929**	0.857	**1.000**	**0.923**
Rule 13 Section	0.651	1.000	0.400	0.571	**0.860**	1.000	**0.760**	**0.864**
Rule 14 Section	0.560	0.714	0.357	0.476	**0.840**	**0.813**	**0.929**	**0.867**
Rule 15 Section	**1.000**	1.000	**1.000**	**1.000**	0.955	1.000	0.933	0.966
Rule 16 Section	**0.941**	1.000	**0.909**	**0.952**	0.882	1.000	0.818	0.900
Rule 17 Section	0.750	1.000	0.647	0.786	0.750	1.000	0.647	0.786
Rule 18 Section	0.718	1.000	0.333	0.500	**0.944**	1.000	**0.867**	**0.929**
Rule 19 Section	0.500	1.000	0.333	0.500	**0.750**	1.000	**0.667**	**0.800**
Rule 20 Section	0.829	0.944	0.739	0.829	**0.902**	**1.000**	**0.826**	**0.905**
Rule 21 Section	0.714	1.000	0.640	0.780	**0.825**	1.000	**0.780**	**0.876**
Rule 22 Section	0.500	0.714	0.333	0.455	**0.917**	1.000	**0.867**	**0.929**

0.153 높았다. 정밀도에서는 0.023으로 매우 비슷한 성능을 보였으며, 두 도구 모두 오탐의 개수가 적음을 확인하였다. CPPCHECK의 재현율은 0.585이고, A 도구는 0.820으로 CPPCHECK의 재현율이 낮아 미탐 개수가 많다는 것을 알 수 있었다. 또한, 규칙을 기준으로 분석한 결과, 두 도구 모두 탐지 못하는 5가지 규칙이 있음 확인하였으며, 추가적인 방법으로 분석을 수행해야 함을 확인하였다. 두 번째로 규칙 카테고리를 기준으로 분석한 결과는 2가지 도구 모두 Advisory, Required, Mandatory 순으로 성능이 좋았으며, 3가지 카테고리에서 모두에서 대체로 A 도구와 CPPCHECK의 평가지표가 높았다. 하지만, 가장 중요한 카테고리인 Mandatory에서 Required, Advisory 보다 성능이 낮음을 확인하였다. Mandatory는 치명적인 오류를 범할 수 있는 규칙 카테고리이므로 추가적인 검토 방법을 고려해야 한다. 마지막인 규칙 섹션을 기준으로 분석한 결과, 14개의 규칙 섹션에서 A 도구이 평가지표에서 대체로 좋은 성능을 보였으며, 4개의 규칙 섹션에서는 2가지 도구의 성능이 비슷하였다. 반면, 3개의 규칙섹션에서는 CPPCHECK가 A 도구보다 좋은 성능을 보였다.

3가지 기준으로 2가지 도구에서 정적분석을 수행한 결과, 공개 소프트웨어 검증 도구인 CPPCHECK보다 상용 도구인 A 도구의 전반적으로 성능지표가 좋았음을 확인하였다. 하지만, 9, 15, 16 규칙 섹션에서 CPPCHECK가 좋은 성능을 보이기도 한 만큼 CPPCHECK의 활용가능성을 확인할 수 있었다. 또한, 탐지하지 못하는 규칙도 있어 도구 이외도 추가적인 방법을 고려해야 함을 확인하였다.

본 연구는 MISRA C:2012 예제 모음을 이용하여 공개 및 상용 정적분석 도구들의 성능을 비교·분석하였다. 그러나 MISRA C:2012는 C90/99 문법을 기준으로 작성된 코딩 규칙으로 이후 등장한 C언어 컴파일러 및 언어 기능 변화에 맞지 않을 수 있다. 최근에는 C11, C18 등 최신 문법을 적용한 개발이 주를 이루고 있어, 이러한 모던 C 코드에 대해 MISRA C:2012만으로 안정성을 평가하기 어려울 수 있다. 이를 보완하기 위해 C11/18 문법 및 기능까지 통합한 MISRA C:2023이 발표되었다[1]. 이에 따라 MISRA C:2023 규칙 지원을 포함하도록 공개·상용 정적분석 도구들이 개발 및 업데이트되고 있다. 따라서 향후 연구에서는 MISRA C:2023 규칙을 지원하는 정적분석 도구들을 성능을 비교 분석할 필요가 있다.

REFERENCES

[1] MISRA C/C++, *https://misra.org.uk*.

[2] Common Weakness Enumeration(CWE), *https://cwe.mitre.org*.

[3] Y. Kim, H. Yoon, “Research on the Effects of MAAB Style Guidelines for Weapon System Embedded Software Reliability Improvement,” Journal of the Korea Institute of Military Science

www.kci.go.kr

and Technology, Vol. 17, No. 2, pp. 213-222, April 2014, doi: 10.9766/KIMST.2014.17.2.213.

[4] S. Lipp, S. Banescu, A. pretschner, “An Empirical Study on the Effectiveness of Static C Code Analyzers for Vulnerability detection,” Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 544-555, 2022, doi:10.1145/3533767.3534380.

[5] RATS, *https://code.google.com/p/rough-auditing-tool-for-security.*

[6] Flawfinder, *https://www.dwheeler.com/flawfinder.*

[7] CPPCHCEK, *https://cppcheck.sourceforge.io.*

[8] A. Kaur, R. Nayyar, “A Comparative Study of Static Code Analysis Tools for Vulnerability Detection in C/C++ and Java Source Code,” Procedia Computer Science, Vol. 171, pp. 2023-2029, 2020, doi:10.1016/j.procs.2020.04.217.

[9] H. Kaur, P. Jai, “Comparing Detection Ratio of Three Static Analysis Tools,” International Journal of Computer Application, Vol. 124, No.13, pp. 35-40, August 2015, doi:10.5120/ijca2015905749.

[10] G. R. Wang, P. S. Chen, “A GCC-based Compliance Checker for Single-translation-unit, Identifier-related MISRA-C Rules,” ICPP Workshops, No. 6, pp. 1-4, 2020, doi:10.1145/3409390.3409396.

[11] C. Y. Chen, Y. A. Chih-Yuan, G. R. Wang, P. S. Chen, “A GCC-based checker for compliance with MISRA-C’s single-translation-unit rules,” Connection Science, Vol. 35, No. 1, pp.1-4, June 2023, doi:10.1080/09540091.2023.2222934.

[12] MISRA C:2012 Example Suite, *https://gitlab.com/MISRA/MISRA-C/MISRA-C-2012/Example-Suite.*

[13] Juliet Test Suite, *https://samate.nist.gov/SARD/test-suites/112.*

[14] R. Amankwah, J. Chen, H. Song, and P. K. Kudjo, “Bug detection in Java code: An extensive evaluation of static analysis tools using juliet testsuites,” Software: Practice and Experience, Vol. 53, No. 5, pp. 1125−1143, December 2022, doi:10.1002/spe.3181.

[15] M. Alqaradaghi, T. Kozsik, “Comprehensive evaluation of static analysis tools for their performance in finding vulnerabilities in Java code,” IEEE Access, Vol. 12, pp. 55824-55842, March 2024, doi:10.1109/ACCESS.2024.3389955.

[16] S. Anand, C. S. Păsăreanu, W. Visser, “JPF−SE: A symbolic execution extension to Java Pathfinder,” Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 134-138, March 2007.

[17] C. L. Blackmon, D. F. Sang, C. S. Peng, “Performance evaluation of automated static analysis tools,” GSTF Journal on Computing (JoC), Vol. 2, No. 1, pp. 1-7, 2014.

[18] N. Visalli, L. Deng, A. Al-Suwaida, Z. Brown, M. Joshi, B. Wei, “Towards automated security vulnerability and software defect localization,” Proceedings of the 2019 IEEE 17th International Conference on Software Engineering Research, Management and Applications (SERA), pp. 90-93, May 2019, doi: 10.1109/SERA.2019.8886795.

[19] Infer, *https://fbinfer.com.*

[20] SonarQube, *https://www.sonarqube.org.*

[21] SpotBugs, *https://spotbugs.github.io.*

[22] Find Security Bugs, *https://find-sec-bugs.github.io.*

[23] PMD, *https://pmd.github.io.*

[24] The open standards for drone hardware, *https://pixhawk.org.*

[25] J. Jang, Y. Kang, J. Lee, “Static Analysis and Improvement Opportunities for open source of UAV flight control software,” Journal of the Korean Society for Aeronautical & Space Sciences, Vol. 49, No. 6, pp. 473-480, June 2021, doi:10.5139/JKSAS.2021.49.6.473.

[26] D. Jung, S. Ahn, J. Choi, “A Programming Enhancements for Embedded Software Development – focus on MISRA-C,” Journal of KIISE: Computing Practices and Letter, Vol. 19, No. 3, pp. 149-152, March 2013.

[27] M. Jeong, C. Moon, H. Ryu, J. Kim, “Development of Automotive Active Control Engine Mount Software using for Static Analysis,” KSAE Annual Spring Conference Proceedings, pp. 1344-1348, 2016.

[28] J. Song, R. Shigdel, A. Pokharel, J. Alves-Foss, “Real-Time Operating Systems’ Compliance with MISRA C Coding Standard: A Comprehensive Study,” IEEE Access, Vol. 12, pp. 10894-10910, October 2024, doi:10.1109/ACCESS.2024.3479971.

[29] T. Charest, N. Rodgers, Y. Wu, “Comparison of static analysis tools for Java using the Juliet test suite,” 11th International Conference on Cyber Warfare and Security, pp. 431-438, 2016.

[30] J. D. A. Pereira, M. Vieira, “On the use of open-source C/C++ static analysis in large projects,” 16th European Dependable Computing Conference(EDCC), pp. 97-102, 2020, doi:10.1109/EDCC51268.2020.00025.

Authors

Donghee Ha received the B.S., M.S. in Computer Science and Engineering from Chungnam National University, Korea, in 2018 and 2020. He is currently a researcher at the Agency for Defense Development.

He is interested in Software Testing, Software Testing for AI, AI for Software Testing.

www.kci.go.kr