情報保護學會論文誌
第 23 卷 第 2 號, 2013. 4

시큐어 코딩 중심으로 본 원자력 관련 소프트웨어*

정 다 혜,† 최 진 영,‡ 이 송 희
고려대학교

Nuclear-related Software analysis based on secure coding*

Da-Hye Jung,† Jin-young Choi,‡ Song-Hee Lee
Korea university

요 약

스마트 시대로 진입하면서, 다양한 임베디드 소프트웨어, 특히 SCADA 소프트웨어와 자동차 소프트웨어 등은 신뢰도와 고 안전성뿐만이 아니라 높은 보안성도 중요하게 되었다. 따라서 해커가 공격하는 데 사용하는 소프트웨어 취약점(vulnerability)의 근본 원인인 소프트웨어 보안 약점(weakness)을 개발 단계에서 제거하는 것이 매우 중요하게 되었다. 기능성 중심의 MISRA-C와 같은 코딩 룰은 보안성 중심의 시큐어 코딩규칙으로 확대가 될 필요가 있다. 본 논문에서는 고 안전성 소프트웨어의 데모용으로 개발 중인 원자력 관련 소프트웨어를 CERT-C 시큐어 코딩 규칙으로 조사하여 얼마나 많은 보안약점을 내재하고 있는 지를 분석하여, 이러한 보안약점을 소프트웨어 개발 시에 제거하는 방법에 대하여 제안한다.

ABSTRACT

We have entered into an era of smart software system where the many kinds of embedded software, especially SCADA and Automotive software not only require high reliability and safety but also high-security. Removing software weakness during the software development lifecycle is very important because hackers exploit weaknesses which are source of software vulnerabilities when attacking a system. Therefore the coding rule as like core functions of MISRA-C should expand their coding focus on security. In this paper, we used CERT-C secure coding rules for nuclear-related software being developed to demonstrate high-safety software, and proposed how to remove software weakness during development.

Keywords: embedded software, SCADA, software vulnerabilities, software weakness, secure coding

I. 서 론

ISC2의 보고에 의하면 해킹의 75%이상이 응용프로그램 내에 존재하는 보안 약점을 악용하여 공격하는 것으로 발표되고 있다[1]. 이처럼 보안 약점(weakness)은 해커들의 공격 시 목표가 되며 프로그램의 오류/에러등 보안 취약점(vulnerability)을 유발시킨다. 따라서 고 신뢰성과 고품질이 요구되는 자동차, 우주항공, 원자력 등의 분야에서는 반드시 보안 약점을 사전에 제거할 필요가 있다. 왜냐하면 이들 시스템에서의 보안 사고는 막대한 재산상의 피해뿐만아니라 인명사고로 이어질 수도 있기 때문이다.

그 예로 최근 망분리가 되어있는 이란 부셰르 원전 핵발전소의 스턱스넷(stuxnet)공격을 들 수 있다.

접수일(2012년 9월 12일), 수정일(1차: 2012년 10월 18일, 2차: 2012년 11월 29일), 게재확정일(2013년 1월 31일)
* 본 논문은 2012년 한국인터넷진흥원 '시큐어코딩 기반 SW 개발보안 기반기술 연구' 위탁과제의 연구결과로 수행되었음(KISA-2012-024).
* 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임 (2012R1A1A2009354)
† 주저자, dhjung@formal.korea.ac.kr
‡ 교신저자, choi@formal.korea.ac.kr

스턱스넷은 슈퍼 산업시설 바이러스로 폐쇄망으로 운영되는 원자력, 전기, 철강 등의 주요기반시설의 제어 시스템에 침투하여 오동작을 유도하는 명령코드를 입력하여 시스템을 마비시킨다. 알려진 바로 독일 지멘스사의 산업자동화제어시스템(PCS7)을 주공격 목표로 제작되었다. 스턱스넷은 Windows 바로가기의 취약성 MS10-046등 당시 알려진 1개의 취약점과 다수의 제로데이 취약점이 활용되어 만들어 졌다. 이란의 스턱스넷 공격은 USB를 통하여 감염되었으며 20%의 원심분리기를 일시 가동 중단 시켰고, 원자력 프로그램을 지연시켰다[2]. 이러한 사건은 보안의 영역이 아니라고 지금까지 생각이 되었던, SCADA 시스템 및 자동차, 항공기 시스템도 해커의 공격에 취약하다는 것을 단적으로 보여준 예라고 할 수 있다.

원격자동제어시스템 (SCADA)처럼 한 번 이라도 다운이 되면 사회적 큰 파장을 일으키는 은행의 온라인 시스템이나 철도, 항공기 운행, 제어 시스템인 고임무 시스템 (mission critical system)과 시스템의 장애가 사람에게 치명적인 피해를 줄 수 있는 항공기, 원자력 발전 제어 설비, 의료 설비 등 사람의 생명과 밀접한 관계가 있는 시스템인 고 안전 시스템(safety-critical system)에서는 보안취약점이 존재해서는 안 된다.

시큐어 코딩은 소프트웨어 개발 단계 (Software Development Life Cycle : SDLC)에서 보안 약점을 제거함으로서 소프트웨어의 취약점과 해킹의 위험성을 줄여주는 방어적 프로그래밍 기법이다[3]. 또한 NIST에서 발표한 내용에 따르면 개발완료단계에서의 보안 약점을 제거하는 비용은 설계단계에서 보안 약점을 제거하는 비용보다 30배정도 더 필요하다고 한다[4]. 즉, 시큐어 코딩은 보안성 강화 뿐만 아니라 비용적인 측면에서도 강점을 갖고 있다. 이에 행정안전부에서는 2012년 12월부터 'SW개발보안제도'를 의무적으로 시행하도록 제도화함으로써 안전한 소프트웨어를 개발하여 각종 사이버 위협으로부터 예방, 대응하고자 노력하고 있으며[5], 미국에서는2002년부터 이미 연방정보보안관리법 (FISMA)를 제정하여 '시큐어코딩'을 의무화하고 있다[6].

본 논문에서는 시큐어 코딩의 필요성을 인식하고 ISO/IEC 9999:1999 기반으로 작성된 C 코드 규칙인 CERT C 시큐어 코딩 규칙을 원자력 발전소에서 활용 될 수 있는 감시 시스템(가칭)에 적용하여 시스템 내에 존재하는 보안 약점에 대하여 분석하고 이에 대한 약점을 소프트웨어 개발 시에 제거하는 방법에 대하여 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 시큐어 코딩, 그리고 분석 방법과 도구를 소개한다. 3장에서는 사례 연구를 통하여 문제점과 원인을 분석하고 4장에서 결론을 맺는다.

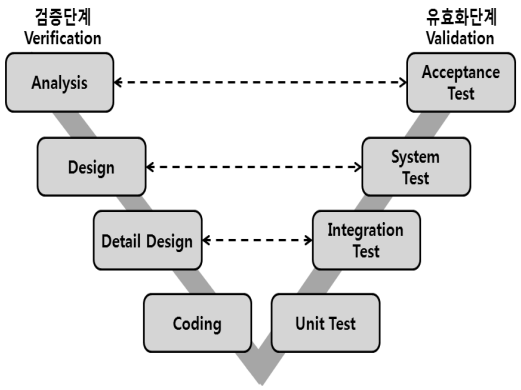
II. 관련연구

2.1 시큐어 코딩

시큐어 소프트웨어(Secure Software)는 보안 약점이 없는 소프트웨어를 의미한다. 물론 보안 약점이란 계속 발견될 수 있으므로 모든 보안 약점을 제거한다는 것은 사실 불가능하다. 그렇지만, 최대한으로 보안 약점이 제거가 된 소프트웨어를 개발하는 것을 목적으로 한다. 시큐어 코딩은 시큐어 소프트웨어를 만드는 방어적 프로그래밍 기법이다. 시큐어 코딩 규칙은 애플의 아이폰 시큐어코딩 규칙처럼 기업에서 만들어 배포하는 경우, MISRA-C 처럼 자동차 협회에서 만들어 배포하는 경우, 아니면 개인적으로 만들어 배포하는 경우가 있다. 그리고 시큐어 코딩 규칙의 적용 대상 범위가 특정 범위인 경우가 있는가 하면 모든 분야에 적용 가능하기도 한다. 2004년에 릴리즈된 MISRA-C는 영국 자동차 산업 신뢰성 협회 (MISRA : Motor Industry Software Reliability Association)에서 발표한 C 코딩 가이드라인이다. 121개의 필수 규칙과 20개의 권고 규칙이 있으며 자동차, 우주항공, 통신, 의료, 국방은 제조분야에 적용가능하다.

CERT C는 미국 정부에서 지원하는 카네기 멜론 대학 내의 Software Engineering Institute에서 개발 하였고 89개의 필수 규칙과 132개의 권장 규칙이 있으며 제조분야뿐만 아니라 C언어로 작성된 모든 소프트웨어에도 적용할 수 있다. 그리고 소프트웨어 오류, 보안 결함, 소프트웨어 취약점으로 이어질 수 있는 C언어의 일반적인 에러/오류를 열거하고 어떻게 안전한 코드를 작성 할 수 있는지를 제안하며 호환 가능하지 않은 코드를 구별하고 있다[7].

[그림 1]은 소프트웨어 개발 프로세스의 한 종류인 V모델이다. V모델의 왼쪽 검증단계부분에서는 정적 분석을 오른쪽 유효화 단계에서는 동적 분석을 한다. 정적분석과 동적 분석은 본 논문에서 사용 한 LDRA 같은 도구를 이용 할 수 있다. V 모델은 요구사항 분석(Analysis)부터 테스트에 대한 문서를 작성하게



〔그림 1〕 소프트웨어 개발 프로세스의 V모델

된다. 코딩(Coding)에서는 이러한 문서를 바탕으로 구현하게 된다. 그리고 단위 테스트(Unit Test)는 시큐어 코딩 규칙이 올바르게 준수되는지에 대한 분석을 하게 된다. 이와 같이 개발 단계별로 코딩 규칙에 대한 테스트 문서 작성이나 검사를 함으로써 사후조치 비용 절감과 소프트웨어 보안 약점을 사전에 제거할 수 있어 보안 취약점 감소 효과를 기대 할 수 있다.

〔표 1〕 임베디드 소프트웨어의 코딩 규칙

코딩 규칙	분 야
MISRA-C : 2004	제조 분야 (자동차, 우주 항공, 통신, 의료, 국방) 주로 차량용 소프트웨어 신뢰성 향상을 목적
JSF(Joint Strike Fighter)Air vehicle C++ coding standard〔8〕	전투기, 폭격기, 지상공격기 주로 미영 항공기 소프트웨어 신뢰성 향상을 목적
CERT C	C언어로 작성된 모든 소프트웨어에 적용 가능

2.2 분석 방법 및 도구

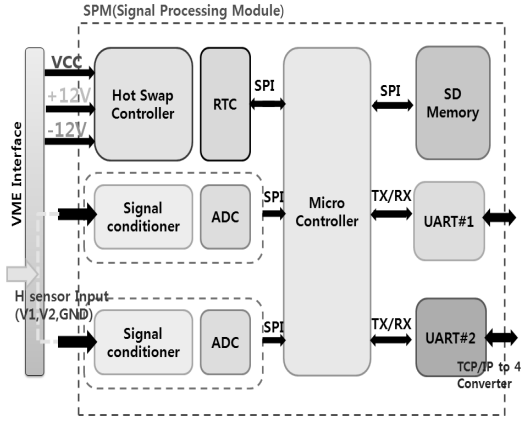
소스 코드 분석 방법에는 동적 분석 (dynamic analysis)과 정적 분석 (static analysis) 방법으로 분류할 수 있다. 동적 분석은 실제 프로그램을 테스트 데이터에 의해 실행하는 방식이고 본 논문에서 사용한 정적 분석은 프로그램 실행을 하지 않고 소스코드로 테스트하는 방법이다. 정적분석을 위해 사용가능한 도구로는 CodeCheck, Panorama C/C++, Compass/Ross, LDRA 등이 있다. 본 논문의 사례 연구에서는 CERT C 시큐어 코딩 규칙을 LDRA TestBed로 정적분석을 수행하고 LDRA TBvision으로 코드 리뷰를 하였다. LDRA는 호스트(host)와 임베디드 소프트웨어에 대해 정적 코어(core static)와 동적 분석 엔진을 제공한다. 그 중 LDRA TestBed는 소스코드 테스팅과 소프트웨어의 확인과 검증 (validation and verification) 분석을 제공한다. LDRA TBvision은 소스코드의 보안 약점과 결함 및 품질 표준 요구사항에 대한 만족 결과와 코드 리뷰를 제공 한다.

III. CERT C 시큐어 코딩 규칙 적용 사례 연구

3.1 분석 대상

원자력 발전소에서 활용 될 수 있는 감시 시스템(가칭)은 소프트웨어의 품질과 안전성이 필수적인 시스템이다. 감시 시스템은 원자로의 중대 사고를 방지하기위해 수소농도를 실시간 모니터링 하는 목적으로 사용되며 감시 시스템의 소프트웨어 오류는 막대한 피해를 유발할 수 있다. 그러므로 소프트웨어의 고품질, 고안전성 그리고 높은 신뢰성이 필수적인 시스템이다.

실험 대상은 감시 시스템의 신호 처리 모듈 (Signal Processing Module)이다. 신호 처리 모듈은 데이터 처리 장치 내에 확장 가능한 형태의 독립된 신호 처리 장치로서 수소 센서 모듈로부터 출력되는 수소농도에 비례한 기전력을 측정하고 분석하여 수소농도 변화를 측정하는 장치이다. 신호 처리 모듈의 전반적인 제어와 계측을 담당하는 중앙처리장치는 〔그림 2〕에서 보이는 Micro Controller이다. Hot Swap



〔그림 2〕 신호처리모듈 (Signal Processing Module, SPM)의 구성도

controller는 신호 처리 모듈의 고장 시 데이터 처리 장치의 운영 상태에서 모듈 교체가 가능하도록 하기 위한 전원안정화 장치이다.

실험 환경으로는 LDRA Testbed 9.1.0을 이용하여 CERT C 시큐어 코딩 규칙을 C언어로 작성된 신호처리 모듈 코드에 실험하였다. LDRA Testbed 9.1.0 도구에서는 CERT C의 68개 필수 규칙과 108개 권고 규칙을 사용하였다.

3.2 분석 결과

감시 시스템의 신호 처리 모듈(Signal Processing Module)을 Cert C 시큐어 코딩 규칙에 적용한 결과는 표2와 같다.

〔표 2〕를 통해서 신호 처리 모듈의 CERT C 시큐어 코딩 규칙 위반 결과를 알 수 있다. 표 2의 cert code는 CERT 시큐어 코딩 규칙을 분류한 이름이다. 권교 규칙과 필수 규칙에 표시된 숫자는 각 규칙 별로 위반 된 개수를 명시하고 있다. 예로 Preprocessor

〔표 2〕 신호처리모듈의 CERT C 시큐어코딩 규칙 위반 결과

cert code	권고 규칙 (Recommendation)	필수 규칙 (Rule)
preprocessor (PRE)	100	0
declarations and initialization(DCL)	211	22
expressions (EXP)	29	465
integers(INT)	826	119
floating point (FLP)	0	2
arrays (APR)	0	0
characters and strings(STR)	0	0
memory management (MEM)	0	0
Input output (FIO)	0	0
environment (ENV)	0	0
signals (SIG)	0	0
error handling (EPR)	0	0
application programming interfaces (API)	0	0
concurrency (CON)	0	0
miscellaneous (MSC)	255	0
posix (POS)	0	0
EXP+FIO+PRE	0	75
DCL + EXP	519	9
INT+ STR	0	214
FLP + INT	0	2
ERR +EXP	0	10

는 권고 규칙에서 100번을 위반하고 필수 규칙은 0번 위반하였다. 표 아래 부분에 EXP+FIO+PRE의 의미는 3개의 부분에서 동시에 위반하였다는 의미이다.

〔표 3〕 신호처리모듈의 CWE Code 규칙 위반 결과

CWE Code	위반개수	CWE Code	위반개수
190	2	783	10,
192	2	192, 197	155
195	478	398, 747	77
252	1	190, 680, 681	66
398	43	192, 197 ,758	37
457	88	192, 194, 197	13
478	1	252, 273, 737	6
483	28	252, 273, 391	6
547	100	561, 570, 571	1
561	9, 9	737, 758, 768	1
563	157	192, 195, 196, 197	214
606	4		
628	38	457, 665, 758, 824	10
682	196		
697	11	664, 682, 737, 747, 768	13
704	1		
735	4	685	5
737	31		
758	352		

〔표 2〕의 결과를 통해 권고 규칙에서는 정수(Integers)부분, 필수 규칙에서는 표현식 (expressions) 부분에서 위반이 많았다. 〔표 3〕은 신호 처리 모듈에 CWE(common weakness enumeration) 규칙을 적용하여 위반한 개수를 명시하였다. CWE는 공개된 보안 취약성에 대한 이름의 표준화를 추진하는 CVE(Common Vulnerabilities and Exposures)를 관리하는 MITRE이 프로젝트이다. 미국 국토안보부내 국가 사이버 보안국의 지원 아래에 MITRE는 일반적으로 널리 통용되는 소프트웨어의 주요 취약점과 보안상의 문제들을 다양한 관점에서 분류 하였다. CWE Code 는 CWE 규칙 번호이다. 예로 표2의 일부를 해석하면 CWE Code의 190번 규칙은 신호 처리 모듈에서 2번 위반되었다. 그리고 190번, 680번, 그리고 681번의 규칙은 동시에 66번 위반되었다.

사례연구는 위의 결과와 2011 CWE/SANS의 매우 위험한 25가지 프로그래밍 오류를 바탕으로 분석하였다[9].

3.1.1 CERT INT13-C와 CWE195

〔표 4〕 신호처리모듈의 CERT C 시큐어 코딩 위반 결과

위반 개수	LDRA 코드	필수 규칙	CERT 코드	CWE
478	331S	비트 연산자는 unsigned 피연산자에만 사용하라.	CERT INT13-C	CWE 195

〔표 5〕CERT C 시큐어 코딩 INT13-C 위험 평가

제안	심각도	발생 가능성	개선비용	우선순위	레벨
INT13-C	높음	낮음	보통	P6	L2

〔표 6〕 CWE 195

CWE	필수 규칙
CWE 195	signed에서 unsigned로의 변환 에러

〔표 7〕 LDRA 331S

LDRA 코드	필수 규칙
LDRA 331S	unsigned형의 모든 상수에는 접미사 'U'를 사용하여야 한다. (Literal value requires a U suffix)

〔표 2〕의 결과를 통해 권고 규칙에서는 정수(Integers)부분 위반이 많았다. LDRA를 통해 정적분석을 하면 〔표 4〕처럼 관련된 LDRA코드 규칙과 CWE 규칙을 명시해준다. INT-13C는 LDRA 코드 규칙의 50S, 120S, 그리고 331S와 관련이 있다. 그러므로 정적분석결과 INT-13C의 규칙을 위반하였다고 해도 각 각 다른 이유로 적혀있다. 본 논문에서 실험한 신호 처리 모듈에서는 INT-13C의 LDRA 331S규칙을 478번 위반한 것으로 나타났다. 〔표 5〕를 통해 알 수 있듯이 INT13-C는 발생가능성은 낮지만 심각도는 높고 개선비용은 보통으로 측정 되었다. CERT INT13-C는 "비트 연산자는 unsigned 피연산자에만 사용하라."라고 명시되어 있고 CWE 195에는 "signed에서 unsigned로의 변환 에러"라 명시 되어 있다. 이 두 규칙은 unsigned에 대한 규칙이고 LDRA의 331S 규칙에는 "unsigned형의 모든 상수에는 접미사 'U'를 사용하여야 한다." 라고 명시되어 있다.

〔표 8〕은 바른 접미사 사용법에 대해 설명하고 있다. 그리고 〔표 9〕를 통해 신호 처리 모듈에서 위반된 사례를 보여준다. 감시 시스템의 신호 처리 모듈에서는 unsigned형을 많이 사용하였지만 접미사 U를 사용한 곳은 없다. 예로 PaketDecoder함수에서 ReturnCmd=0; 이라 선언 할 때 ReturnCmd 변수는 unsigned로 선언되어 있기에 ReturnCmd=0U; 로 수정되어야 한다. 수치 상수를 사용하지 않은 코드는 C언어의 컴파일러가 자동적으로 치환문의 왼쪽 변수 타입에 따라 치환문 오른쪽에 있는 타입을 변환한다. 만약 잠정적으로 결정된 타입을 수치 상수에 사용하기 원하지 않는 다면 접미사를 사용하여 정확한 타입을 지정해 주면 된다. 만약 접미사를 사용한 정수타입 상수가 오버플로가 발생하면 컴파일러는 경고를 준다. 그러나 접미사를 사용하지 않은 불명확한 설명의 정수형 상수의 경우는 프로그램의 의도한 값이 아닌 가장 작은 값이 나올 가능성이 있다〔10〕. 감시 시스템은 정확한 값을 전달하여 위험을 미리 방지하는 것이 목적이다. 그러므로 접미사를 사용하여 의도하지 않는 값이 나올 가능성을 사전에 차단하여야 한다.

〔표 8〕 접미사 U의 올바른 사용법

```
… u8a +   5          /*규칙 준수 되지 않음*/
… u8a +   5U         /*규칙 준수*/
… u8a +   (unit8_t)5   /*규칙 준수*/
```

〔표 9〕 신호 처리 모듈 예제

부적절한 코드

```
Unsigned char   PaketDecoder(unsigned char ptnum)
{
    Volatile unsigned char  ReturnCmd;
    ReturnCmd = 0;
                    …
}
```

해결 후

```
Unsigned char PaketDecoder(unsigned char ptnum)
{
    Volatile unsigned char ReturnCmd;
    ReturnCmd=0U;
                    …
}
```

〔표 10〕 CVE(Common Vulnerabilities and Exposures)의 CWE 195 사례

CVE	설 명
CVE-2007-4268	Apple Mac OS X 10.4부터 10.4.10 버전은 AppleTalk 메시지를 통해 임의의 코드가 로컬 사용자에 의해 실행되었다.

〔표 10〕의 사례는 CWE195 규칙 위반을 통해 실제로 일어난 정수형 signedness 에러 사례이다. signedness 에러는 signed형이 unsigned형으로 인식되거나 unsigned형이 signed형으로 인식되면서 생기는 오류이다. signed형 에러는 정수형 오버플로에 의해 일어나며 정수형 오버플로는 의도적인 버퍼 오버플로를 발생 시킬 수 있다.

3.3 정적 분석의 한계

임베디드 소프트웨어는 하드웨어에 의존적인 부분이 많다. 그 이유 때문에 정적 분석 시 모든 펌웨어를 포함 시켜 주지 않을 경우 거짓 양성 (false positive: 실제로는 문제가 없음에도 불구하고 문제로 인식되는) 경보가 울리기도 한다.

3.3.1 거짓 양성 경보 사례 : CERT EXP37-C와 CWE628

〔표 11〕 신호처리모듈의 CERT C 시큐어 코딩 위반 결과

위반 개수	LDRA 코드	필수 규칙	CERT 코드	CWE
9	496 S	API에 의해 의도된 인자들로 함수를 호출하라	CERT EXP37-C	CWE 628

〔표 12〕 CERT C 시큐어 코딩 EXP37-C 위험평가

제안	심각도	발생가능성	개선비용	우선순위	레벨
EXP37-C	보통	약간 높음	높음	P4	L3

〔표 13〕 CWE 628

CWE	필수 규칙
CWE 628	부정확한 인자를 통한 함수 호출

〔표 14〕 LDRA 496S

LDRA 코드	필수 규칙
LDRA 496S	선언되어 있지 않는 함수 호출(Function call with no prior declaration)

CERT EXP37-C는 "API에 의해 의도된 인자들로 함수를 호출하라."이다. 이 규칙은 〔표 11〕을 통해 알 수 있듯이 CWE 628(Common Weakness Enumeration)의 "부정확한 인자를 통한 함수 호출"과 관련 있다. 함수 호출시 부정확한 인자나 인자타입을 사용하면 기대하지 않은 프로그램 수행을 유도하게 된다. API 사용 시 이러한 상황이 생기면 컴파일러는 경고 메시지만 출력한다. EXP37-C는 심각도는 보통이지만 발생가능성이 약간 높고 개선 시 비용이 많이 든다.

〔표 15〕 CERT C 사용 예제

부적절한 코드

```
#include <stdio.h>
#include <string.h>

char *(*fp) ();

int main(void) {
  char  *c;
  fp = strchr;
  c = fp(12,2);
  printf("%s\n", c);
}
```

해결 후

```
#include <string.h>

char *(*fp) (const char *, int);

int main(void) {
  char  *c;
  fp = strchr;
  c = fp(12,2);
  printf("%s\n", c);
}
```

〔표 15〕에서는 EXP37-C 규칙의 위반과 올바른 수정에 대한 예제이다. 〔표 15〕의 부적절한 코드 부분에서 fp가 함수 프로토타입 없이 선언되었기에 fp(12,2)가 호출 되었을 때 아무 타입 체크도 일어나지 않는다. 그러므로 fp를 정확한 함수 프로토타입으로 선언해주어야 한다.

〔표 16〕은 신호처리 모듈에서 오류라 판정났던 코드 부분이다. 도구에서 pgm_read_float 함수가 정확히 정의되지 않았다고 인식하여 생긴 오류이다. 그러나 감시 시스템의 원래 소스코드에서는 #include <avr/pgmspace.h>에 pgm_read_float의 정의인 #define pgm_read_float (address_short) pgm_read_byte_near (address_short)가 포함되어 있어 함수 선언 문제가 없다. 이와 같이 오류 판정이 난 이유는 정적 분석 시 모든 펌웨어를 포함시키지 않았기에 때문이다.

〔표 16〕 신호 처리 모듈 CERT C 적용 예제

위반 사례

```
float TempSearch(float Voltdata)
{
          /* … */

  for(int i = 0; i<11; i++)
  {
    dreadData                                    =
pgm_read_float(temptable[TempSerCount]   )   *
95.5/1000.0;
   if(dreadData > Voltdata){
TempSerCount = TempSerCount-addCount;
   }
   else{
TempSerCount = TempSerCount +addCount;
       }
     addCount = addCount/2;
  }
    /*…*/
}
```

〔표 17〕 CVE(Common Vulnerabilities and Exposures)의 CWE 628 사례

CVE	설 명
CVE-2006-7049	1.1.6.2 버전 전의 Wikka Wiki에 공격자가 보안 제한을 무시하고 임의적인 PHP script를 접근 가능하게 하였다.

〔표 17〕은 EXP37-C와 관련있는 CWE 628규칙을 위반하여 생긴 실제 사례이다. 사례는 부정확한 인자순서로 strstr과 strrpos 함수를 호출하였기에 일어났다. 이 문제는 코드 감시도구와 잠재적 위반사항을 찾아 낼 수 있는 컴파일러 사용으로 쉽게 예방할 수 있다.

IV. 결론 및 향후 연구

본 논문에서는 고 신뢰성과 고 품질이 요구되는 원자력의 감시 시스템을 대상으로 CERT C의 시큐어 코딩 규칙으로 정적 분석을 수행하였다. 그리고 정적 분석 결과를 통하여 빈번히 일어나는 오류와 CVE(Common Vulnerabilities and Exposures)를 통한 실제 오류 사례에 대해 분석하였다〔11〕. 이러한 오류들은 정적 분석 도구를 사용하면 소프트웨어 개발 단계에서 사전에 쉽게 제거해 줄 수 있어 소프트웨어 보안 취약성과 약점을 사전에 예방 할 수 있다. 그러므로 프로그래머들에게 분석 도구를 이용한 코딩 규칙 준수를 권장한다. 그리고 본 논문의 사례연구를 통해 알 수 있듯이 코딩 규칙은 임베디드 소프트웨어 특성에 따라서 차이가 존재한다. 같은 프로그래밍 언어로 제작되어있다고 해서 소프트웨어의 특성과 맞지 않는 코딩 규칙을 적용 시키면 소프트웨어의 의도와 다르게 수정될 수 있다. 그러므로 각 임베디드 소프트웨어 분야별로 코딩 규칙이 확립되어야 한다. 특히 원자력의 감시 시스템처럼 보안 사고로 물질적 피해와 더불어 인명 피해까지 불러오는 시스템은 개발 시 MISRA-C같이 기능성 중심의 코딩 규칙뿐 만아니라 CERT C처럼 기능성과 보안성이 포함된 코딩 규칙을 확립하여 적용하여야한다. 향후 연구로는 본 논문에서 다루지 못한 오류들에 대한 문제점과 올바른 해결책을 제시하고, 분야별 임베디드 소프트웨어에 적용 할 수 있는 시큐어 코딩 규칙에 대한 연구를 할 계획이다.

참고문헌

〔1〕 Gartner http://gartner.com Nov. 2005.

〔2〕 Nicolas Falliere, Liam O Murchu, Eric Chien, "W32.Stuxnet Dossier version 1.0," Symantec, September. 2010.

〔3〕 A. John, R. Peter, "Electric Communication Development," Communications of the ACM, 40, pp. 71-79, May. 1997.

〔4〕 NIST, "The Economic Impacts of Inadequate Infrastructure for Software Testing," May. 2002.

〔5〕 행정안전부 http://www.mopas.go.kr

〔6〕 NIST(National Institute of Standards and Technology) http://csrc.nist.gov

〔7〕 로버트 C, 현동성 역. 시코드 저,"버그 없는 안전한 소프트웨어를 위한 CERT C 프로그래밍," 에이콘, pp.26, 2010

〔8〕 Joint Strike Fighter http://www.jsf.mil

〔9〕 Common Weakness Enumeration, "2011 CWE/SANS Top 25 Most Dangerous Software Errors," http://cwe.mitre.org/top25/ Sep 13 2011.

〔10〕 Herbert Schildt 저, 유해영/우진운 역, "알기 쉽게 해설한 C" Second Edition, 이한 출판사, p150, July 30, 2003.

〔11〕 Common Vulnerabilities and Exposures http://cve.mitre.org

〈著者紹介〉

정 다 혜 (Da-Hye Jung) 학생회원
2011년 2월: 성공회대학교 소프트웨어공학과 졸업
2011년 9월~현재: 고려대학교 융합소프트웨어전문대학원 임베디드소프트웨어 석사과정
〈관심분야〉 정형기법, 시큐어 코딩, 프로그래밍 언어

최 진 영 (Jin-young Choi) 종신회원
1982년: 서울대학교 컴퓨터공학과 졸업
1986년: Drexel University Dept. of Mathematics and Computer Science, . 석사.
1993년: University of Pennsylvania Dept. of Computer and Information Science, 박사
1993년~1996년: Research Associate. University of Pennsylvania
1996년~1999년: 고려대학교 컴퓨터학과 조교수
1999년~2004년: 고려대학교 컴퓨터학과 부교수
2004년~현재: 고려대학교 컴퓨터·전파통신공학부 교수
〈관심분야〉 정형기법, 임베디드 실시간 시스템, 프로그래밍 언어, 프로세스 대수, 소프트웨어 공학

이 송 희 (Song-Hee Lee) 학생회원
1998년 2월: 순청향대학교 컴퓨터공학과 졸업
2002년 2월: 순천향대학교 전자상거래학과 석사
2009년 2월: 고려대학교 컴퓨터학과 박사
현재: 고려대학교 연구교수
〈관심분야〉 정형기법, 네트워크 보안, 소프트웨어 보안