임베디드 기기 바이너리 취약점 분석 효율성 제고를 위한 중간어 변환 기술

(Intermediate-Representation Translation Techniques to Improve Vulnerability Analysis Efficiency for Binary Files in Embedded Devices)

정병호*, 김용혁**, 배성일**, 임을규***

(Byeoung Ho Jeoung, Yong Hyuk Kim, Sung il Bae, Eul Gyu Im)

요 약

임베디드 기기는 시퀀스 제어 기능과 수치연산 기능을 활용하여 제어 프로그램에 따라 산업현장의 기기 등 다양한 자동화 시스템에 활용된다. 현재 임베디드 기기는 기업의 산업현장, 원전, 대중교통 같은 국가기반시설에서 제어 시스템으로 활용되고 있다. 따라서 임베디드 기기를 대상으로 하는 공격은 큰 경제적 손실과 사회적 손실을 야기할 수 있다. 임베디드 기기를 대상으로 하는 공격은 대부분 데이터, 코드 변조로서 제어 프로그램을 대상으로 이루어진다. 산업 자동화 임베디드 기기의 제어 프로그램은 일반적인 프로그래밍 언어와 달리 회로 구조를 표현하기 위하여 설계되었고, 대부분의 산업 자동화 제어 프로그램은 그래픽 기반 언어인 LAD로 설계되어있어 정적분석이 용이하지 않다. 이러한 특징으로 인하여 산업 자동화 제어 프로그램에 대한 취약점 분석 및 보안 관련 연구는 정형 검증, 실시간 모니터링 수준에 그친다. 또한 사전에 취약점을 탐지하고 공격에 대한 대비가 가능한 산업 자동화 제어 프로그램 정적분석 연구는 매우 저조한 실정이다. 따라서 본 연구에서는 산업 자동화 임베디드 프로그램에 대한 정적분석 효율성 증대를 위하여 회로 구조를 표현하기 위해 설계된 산업 자동화 제어 프로그램을 논리식으로 표현하기 위한 방법을 제시한다. 또한 다양한 제조사의 산업 자동화 제어 프로그램을 통합적으로 분석하기 위하여 LLVM IR을 활용한 중간어 변환 기술을 제안한다. LLVM IR을 활용함으로서 동적 분석에 대한 통합분석이 가능하다. 본 연구에서는 해당 방법에 대한 검증을 위하여 S 社의 제어 프로그램을 대상으로 하여 논리식 형태의 중간어로 변환하는 프로그램의 시제품을 개발하였다.

■ 중심어 : 산업자동화 ; 임베디드 기기 ; 제어 프로그램 ; 중간어 ;

Abstract

Utilizing sequence control and numerical computing, embedded devices are used in a variety of automated systems, including those at industrial sites, in accordance with their control program. Since embedded devices are used as a control system in corporate industrial complexes, nuclear power plants and public transport infrastructure nowadays, deliberate attacks on them can cause significant economic and social damages. Most attacks aimed at embedded devices are data-coded, code-modulated, and control-programmed. The control programs for industry-automated embedded devices are designed to represent circuit structures, unlike common programming languages, and most industrial automation control programs are designed with a graphical language, LAD, which is difficult to process static analysis. Because of these characteristics, the vulnerability analysis and security related studies for industry automation control programs have only progressed up to the formal verification, real-time monitoring levels. Furthermore, the static analysis of industrial automation control programs, which can detect vulnerabilities in advance and prepare for attacks, stays poorly researched. Therefore, this study suggests a method to present a discussion on an industry automation control program designed to represent the circuit structure to increase the efficiency of static analysis of embedded industrial automation programs. It also proposes a medium term translation technology exploiting LLVM IR to comprehensively analyze the industrial automation control programs of various manufacturers. By using LLVM IR, it is possible to perform integrated analysis on dynamic analysis. In this study, a prototype program that converts to a logical expression type of medium language was developed with regards to the S company's control program in order to verify our method.

■ keywords : Industrial Automated Systems ; Embedded devices ; Controll Logic ; Medium Languege ;

* 학생회원, 한양대학교 컴퓨터소프트웨어학과
** 학생회원, 한양대학교 컴퓨터소프트웨어학과
*** 비회원, 한양대학교 컴퓨터소프트웨어학과

접수일자 : 2018년 03월 11일 게재확정일 : 2018년 03월 26일
수정일자 : 2018년 03월 24일
교신저자 : 임을규 e-mail : lmeg@hanyang.ac.kr

Smart Media Journal / Vol.7, No.1 / ISSN:2287-1322
http://dx.doi.org/10.30693/SMJ.2018.7.1.38

Ⅰ. 서 론

오늘날 산업계에서는 기기 자동화 시스템이 일반화 되어 기업의 공정, 원전 등의 산업시설 뿐만 아니라 대중교통과 같은 사회기간 시설에서도 널리 활용되고 있다. 임베디드 기기는 논리회로와 논리연산, 수치연산으로 구성된 산업자동화 제어 프로그램이라는 언어로 설계된 프로그램에 따라 공장의 자동화 과정 제어에 활용된다. 따라서 산업 자동화 임베디드 기기 또는 산업 자동화 제어 프로그램을 대상으로 하는 공격은 타 공격에 비하여 큰 경제적 손실과 직접적인 사회적 혼란을 야기한다. 2010년 이란의 원자력 발전소를 대상으로 한 스턱스넷 바이러스 공격은 이란 원자력 발전소를 제어하는 임베디드 기기를 감염시켜 산업자동화 제어 프로그램에 대한 변조를 진행하였고, 이는 이란의 원자력 발전소를 마비시키는 결과를 야기하였다[1]. 그러나 현재 산업 자동화 임베디드 기기와 산업자동화 제어 프로그램에 대한 취약점이 다수 알려져 있는 반면, 산업자동화 제어 프로그램의 보안에 대한 인식은 매우 부족한 실정이다. 2016년 독일의 OpenSource Security 社 외 3개 연구소는 BlackHat Asia Conference에서 산업자동화 제어 프로그램 형식의 일종인 ST(Structed Text)를 활용하여 제작된 산업자동화 임베디드 기기 대상 웜을 공개하였다[2]. 해당 바이러스는 로컬 네트워크를 통하여 스스로를 전파시키고, C&C(Command and Control) 서버에 접속하여 물리적 조작과 데이터 변조를 수행한다. 현재 산업 자동화 임베디드 기기 관련 보안 연구는 산업자동화 임베디드 기기의 이상 행동을 감시하는 동적 분석방법에 집중되어 있으며, 정적분석의 경우 대부분 정형 검증에 치중되어 있다. 본 연구에서는 산업자동화 임베디드 기기의 산업자동화 제어 프로그램의 바이너리 파일의 코드를 추출하여 해당 코드를 바탕으로 하여 정적 분석 기반 취약점 탐지를 진행하는 기술을 제안한다. 산업자동화 임베디드 기기의 제어 프로그램 언어는 S 社의 S7 시리즈, M 社의 Melsec 등 다양한 제조사의 기기가 존재하며 각 제조사의 산업자동화 제어 프로그램 구조, 문법, 명령어는 상이하다. 따라서 현재 산업자동화 제어 프로그램에 대한 취약점 분석은 각 제조사에 대하여 개별적으로 분석을 진행할 필요가 있다. 이러한 문제점을 개선하고 취약점 분석을 효율적으로 진행하기 위하여 다양한 제조사의 산업자동화 제어 프로그램을 통합적으로 분석할 필요가 있다. 이를 위하여 다양한 제조사의 산업자동화 제어 프로그램을 중간어로 변환하여 통합적으로 분석하기 위한 방법을 제안한다. 본 연구에서는 중간어로 변환하기 위한 방안으로 산업자동화 제어 프로그램의 바이너리 파일에서 추출한 STL 형식으로 표현된 산업자동화 제어 프로그램 코드를 활용하여 산업자동화 제어 프로그램의 회로구조를 논리식으로 변환하는 방법을 활용한다.

Ⅱ. 본 론

1. 기존 연구 현황

University of illinois at Urbana-Champaign의 Chris Lattner 외 1인은 컴파일러 프레임워크 LLVM(Low Level Virtual Machine)에 대하여 기술하였다[3]. LLVM은 프로그램을 컴파일 타임, 링크 타임, 런타임 상황에서 프로그램의 작성 언어에 독립적으로 최적화를 구현 할 수 있도록 구성된 컴파일러 기반구조이다. LLVM의 핵심은 중간 표현(IR)로 어셈블리어와 비슷한 저급 프로그래밍 언어로서 프론트 엔드만 개발하면 여러 아키텍처를 빌드 할 수 있도록 하였다. IEEE의 Herbert Prahofer 외 3인은 산업자동화 임베디드 기기 프로그래밍 언어에서의 정적 분석 기술을 제안하였다[4]. 해당 연구에서는 프로그램 구조의 패턴 매칭, 제어 흐름 및 데이터 흐름 분석, 호출 그래프 기술 등을 활용하여 산업자동화 임베디드 기기 산업자동화 제어 프로그램을 분석하였다. G. P. G Sandaruwan 외 3인은 다양한 경로의 산업자동화 임베디드 기기에 대한 공격 경로를 분석하여 산업자동화 임베디드 기기 취약점을 찾아내었다[5]. 또한 산업자동화 임베디드 기기 기반 시스템에서의 취약점을 보완하기 위한 솔루션을 제안하였다. University Laboratory in Automated Production Resarch의 S. Lamperiere-Couffin 외 2인은 산업자동화 임베디드 기기 프로그램에 대한 정형 검증 방법에 대하여 기술하였다[6]. 본 연구에서는 정형 검증을 위하여 산업자동화 임베디드 기기 프로그램의 SFC(Sequential Function Chart)를 활용하였다. 해당 연구의 경우, 임베디드 기기 프로그램에 대한 문법 오류 등에 대한 검증 방안으로 취약점 분석에 활용이 어렵다. University of Victoria의 Shahid Alam 외 2인은 중간어를 활용하여 악성코드 탐지를 진행하였다[7]. 해당 논문에서는 opcode에 대한 중간어를 설계하여 변환을 진행하였다. 본 연구에서는 제어 명령어, 수식연산 명령어, 논리 명령어, 네트워크 명령어 등으로 분류하여 중간어를 설계하였다. Software Engineering and Technology labs Infosys Technologies Ltd의 Sravan Kumar R 외 1인은 클라우드 저장소에서의 데이터 무결성 검증 방안을 제안하였다[8]. 해당 연구에서는 데이터를 블록 단위로 구획하여 해쉬값을 추출하여 데이터 무결성을 검증하였다.

2. 산업자동화 제어 프로그램의 개요

가. 산업자동화 제어 프로그램의 기본구조

산업자동화 제어 프로그램은 산업자동화 제어 프로그램에서 활용되는 일종의 프로그래밍 언어이다. 그러나 산업자동화 제어 프로그램은 일반적인 프로그래밍 언어와 달리 회로의 구성 방식을 표현하기 위하여 설계되었다. 산업자동화 제어 프로그램을 구성하는 기본 요소는 회로, 접점, 코일이다. 회로는 입력 된 신호가 지나가는 흐름을 의미하며, 전기 회로의 직렬, 병렬 개념과

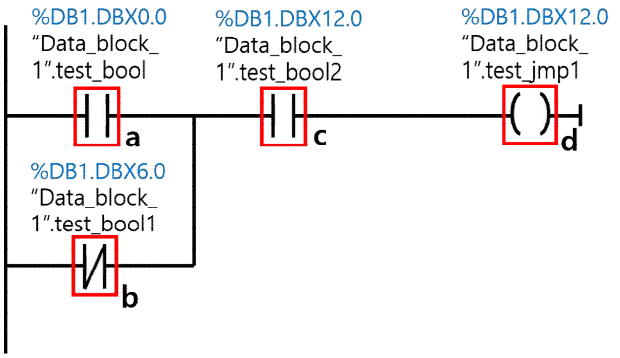

그림 1 LAD 형식 산업자동화 제어 프로그램 코드 예시

상응한다. 접점은 해당 지점을 지나는 신호를 검사하여 통과 여부를 결정한다. 산업자동화 제어 프로그램에서 해당 접점에 신호가 흐르고 있는 경우 1 값을 반환하고, 흐르지 않을 경우, 0 값을 반환한다. 즉, 접점은 현재 상태 조건에 따라 해당 산업자동화 임베디드 기기의 신호를 조건에 따라 흘려보내는 스위치 역할을 수행한다. 코일은 출력으로서 해당 회로의 최종 값을 반영한다. 그림 1은 산업자동화 제어 프로그램의 실제 코드이다. 해당 그림의 a, b ,c은 접점을 의미하며, d는 코일을 의미한다. b는 NOT 게이트 역할을 수행하는 접점이다. 그림 1의 프로그램이 실행 될 경우, 각 시작 지점에서 RLO bit를 활용하여 신호를 전달한다. RLO의 초기값 1이며, 특정 지점에서의 RLO 값은 해당 지점에 신호가 도달하였음을 의미한다. 그림 1은 'Normally Open Contact'이며 참조하는 주소의 데이터 값이 가지는 Boolean 값이 1일 경우 회로의 신호를 통과시킨다. 반면 그림 1의 b 접점은 'Normally Closed Contact'로서 참조하는 주소의 데이터 값이 가지는 Boolean 타입 값이 0일 경우 회로의 신호를 통과시킨다. 예를 들어, 그림 1의 접점 a, b ,c가 참조하는 주소가 가지는 데이터의 값이 모두 1이라고 가정할 때, 1에서는 신호를 통과시키며 해당 부분에서의 RLO의 값은 1이 된다. 반면, b 접점에서는 0일 때 신호를 통과시키므로 신호를 통과시키지 않는다. 따라서 b 접점에서의 RLO 값은 0이 된다. c 접점의 경우, a접점에서 신호를 통과 시켰으므로 a 접점에서 통과시킨 신호를 받는다. c 접점 역시 참조하는 주소가 가지는 데이터의 값이 1이므로 신호를 통과시키고 c접점에서의 RLO 값은 1이 된다. d 코일은 대입 연산자를 의미하며 최종적으로 신호를 받아 해당하는 RLO 비트의 값을 참조한 데이터에 대입한다. 해당 값을 활용하여 산업자동화 임베디드 기기는 조건에 따른 동작을 수행하도록 제어한다.

3. 산업자동화 제어 프로그램 중간어 변환

현재 산업자동화 임베디드 기기의 제어 프로그램 개발 언어는 각 제조사 마다 문법과 명령어가 상이하다. 그러나 산업 현장에서는 필요성에 따라 여러 제조사의 기기를 혼용한다. 또한 각 제조사의 언어와 기기에 따라 개별적으로 관리되고 있다. 따라서 효과적인 취약점 분석을 위해서는 통합 관리 방안이 필요하다. 본 연구에서는 각 제조사의 산업자동화 제어 프로그램 통합 관리를 위하여 중간어 변환 기술을 제안한다.

가. LLVM IR

LLVM IR은 LLVM(Low Level Virtual Machine)의 중간어를 의미한다. LLVM은 다양한 환경에서 프로그램을 실행하기 위한 컴파일러 기반 구조이다. LLVM은 프론트 엔드, 미들 엔드, 백 엔드로 기존 컴파일러의 세 가지 주요 구성요소를 분리한다. 프론트 엔드에서는 C, C++ 등의 언어를 중간어로 변환하기 위한 정의를 포함하고 이를 변환한다. 미들 엔드 단계에서는 최적화를 위한 'LLVM Optimizer'를 제공하고 백 엔드에서 코드 생성기로 변환된 중간어를 기계어로 만들어 컴파일을 진행한다. 본 연구에서는 LLVM의 구동 방식을 차용하여 각 제조사의 명령어를 중간어로 변환하는 프론트 엔드, 취약점 분석을 진행하는 백 엔드로 취약점 분석 프로그램을 구성한다. 또한 LLVM IR의 API와 문법을 활용하여 리스트 형태로 표현되는 중간어를 설계한다. LLVM IR에서 제공하는 API의 활용 예시는 표 1과 같다.

표 2 LLVM IR API

API	Usage	Function
add	add <ty> <op1>, <op2>	<op1> + <op2>
sub	sub <ty> <op1>, <op2>	<op1> − <op2>
mul	mul <ty> <op1>, <op2>	<op1> * <op2>
ret	ret <type> <value> / ret void	<type>의 <value>를 반환
and	and <ty> <op1>, <op2>	<op1> and <op2>
or	or <ty> <op1>, <op2>	<op1> or <op2>
xor	xor <ty> <op1>, <op2>	<op1> xor <op2>

Smart Media Journal / Vol.7, No.1 / ISSN:2287-1322
http://dx.doi.org/10.30693/SMJ.2018.7.1.40

나. 논리식 파스트리 변환

프로그램은 그래픽 기반 언어이므로 중간어 변환에 활용하기에 부적절하다. 따라서 해당 그래픽 기반 프로그램의 정보를 텍스트 형태로 변형할 필요가 있다. STL(Statement List)은 그래픽 기반 언어인 래더 로직을 텍스트 형태로 나타낸 언어이다. STL 형태로 나타낸 산업자동화 임베디드 기기 프로그램의 예시는 다음 그림과 같다. 그림 3은 그림 2와 같이 LAD로 제작된 산업자동화 제어 프로그램 코드를 STL 형태로 변환을 진행한 프로그램 코드이다. 위의 그림과 같이 명령어의 리스트 형태로 나타난다[9]. 그러나 A, O 등의 비트로직 명령어가 일반적인 프로그래밍 언어와 달리 래더 로직에서는 접점의 배치를 의

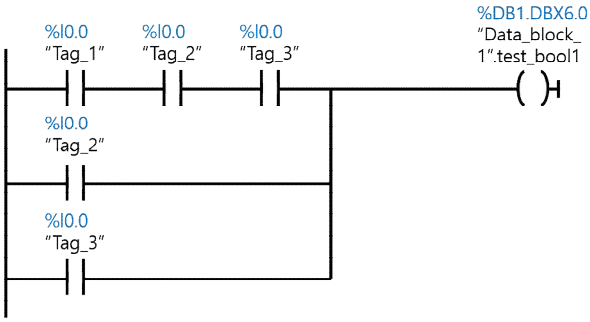

그림 2 LAD 형식 산업자동화 제어 프로그램 코드

```
A       "Tag_1"
A       "Tag_2"
A       "Tag_3"
O       "Tag_2"
O       "Tag_3"
=       "Data_block_1".test_bool1
```

그림 3 STL 형식 산업자동화 제어 프로그램 코드

미하기 때문에 실제 연산 순서를 고려 할 수 있는 장치가 필요하다. STL 명령어 중 접점, 코일, 명령어의 위치를 의미하는 비트로직 명령어는 표 4와 같다. 표 4의 A 연산자가 의미하는 프로그램의 회로 구성은 그림 4와 같다.

STL Program	Relay Logic
Power rail	Power rail
A I 1.0	I1.0 signal state 1 — No Contact
A I 1.1	I1.1 signal state 1 — No Contact
= Q 4.0	Q4.0 signal state 1 — Coil
Displays closed switch	

그림 4 A 명령어 회로도

본 연구에서는 위와 같은 실행 순서와 회로 구조를 반영하기 위하여 중간어 변환 과정에 앞서 파스트리 형태로 변형하는 과정을 진행한다. 본 논문에서는 회로 구조를 논리식 형태로 이루어진 파스트리로 변형을 진행하는 동시에 같은 신호 흐름을 유지하기 위하여 산업자동화 제어 프로그램을 대상으로 변환 규칙을 설계한다. 해당 규칙에 따라 그림 3의 STL 형식의 프로그램은 그림 6의 파스트리와 같은 형태로 변형된다.

표 3 Bit Instruction

Instruction	meaning
A	And : Contact is serise
AN	A Negation
O	Or : Contact is Parallel
ON	Negation of O
A(	A Nested
AN(	A Negation, Nested
O(	O Nested
ON(	O Negation, Nested
)	Close Nest

규칙 1. 노드 이동 규칙
최초로 생성된 노드는 생성 당시, 현재 노드가 되며 새로운 노드가 생성될 경우, 새로 생성된 노드는 현재 노드가 되고 현재 노드였던 노드는 이전 노드가 된다.

규칙 2. Current Instruction = 'A'
명령 A는 1 개의 입력 매개 변수를 가진다. 이전 명령이 'A'이면 입력 매개 변수를 현재 노드의 매개 변수에 추가하고, 그렇지 않으면 새 노드를 생성한다. 새로운 노드가 생성될 경우, 새로 생성된 노드의 연산자는 'and'로 한다. 또한 입력 매개 변수는 새 노드의 매개 변수가 된다. 'and'는 직렬배치를 의미하는 'A'와 달리 논리식의 'and'연산자를 의미한다.

규칙 3. Current Instruction 'A('
명령 'A ('는 입력 매개 변수를 가지지 않는다. 입력 명령이 'A('인 경우, 'and' 연산자를 가지는 새로운 노드를 생성한다. 또한 'A(' 명령어 전 가장 마지막으로 생성된 노드의 연산 결과값은 새로 생성된 노드의 매개 변수가 된다.

규칙 4. Current Instruction '='
명령어 '='는 1개의 입력 매개 변수를 가진다. 명령어 '='는 입력 매개 변수에 해당하는 변수에 대한 회로의 최종 결과값 할당을 의미한다. 이 경우, '='를 가지는 노드를 새로 생성하고 마지막 노드의 연산 결과값을 매개 변수로 추가한다.

규칙 5. Current Instruction 'O('
명령어 'O ('는 입력 매개 변수를 가지지 않는다. 입력 명령이

'O'일 경우, 'and'를 연산자로 가지는 노드를 새로 생성한다. 마지막 노드는 연산 결과값은 새로 생성된 노드의 매개변수가 된다.

규칙 6. Current Instruction 'O'

명령어 'O'는 1 개의 입력 매개 변수를 가진다. 이전 명령이 'O'일 경우, 입력 매개 변수를 현재 노드의 매개 변수에 추가한다. 이전 명령어가 'A(', 일 경우 [or, 입력 매개 변수]의 연산 결과값을 현재 노드의 매개 변수로 추가한다. 그 외의 경우 'or' 연산자를 가지는 새로운 노드를 추가한다. 이 경우, 현재 노드의 결과값과 [or, parameter] 노드의 결과 값을 새로 생성된 노드의 매개 변수로 추가한다.

다. 산업자동화 제어 프로그램 중간어 변환 과정

그림 5는 중간어 변환 과정의 개요를 나타낸다. TiaPortal에서 .awl 형식의 파일에서 STL 형태의 코드를 추출하여 파스트리 생성 모듈에 입력한다. TiaPortal은 S 社의 산업자동화 제어 프로그램 편집, 관리도구이다. 파스트리 생성 모듈에서는 해당 입력을 명령어와 매개 변수로 이루어진 노드로 파싱한다. 이후 이를 순서대로 읽어 들여 파스트리 생성 규칙을 활용하여 Json 형태의 파스트리를 생성한다. 중간어 변환 모듈에서는 해당 파스트리를 입력받아. STL-LLVM IR 명령어 맵핑을 통하여 중간어 변환을 진행한다. 중간어 변환 과정의 입력은 파스트리 형태로 주어진다. 각 파스트리의 노드에는 명령어와 매개 변수 정보가 저장되어있다. 파스트리는 깊이 우선 탐색 방법에 따라 탐색하였을 때, 실제 회로 연산 순서에 맞게 탐색된다. 논리 연산자에서 직렬연산에 해당하는 접점들은 'and' 명령어에 묶이고 병렬연산에 해당하는 연산자는 'or' 명령어로 묶인다. 또한 상위 노드의 비트 연사자의 매개 변수로 하위노드의 연산 결과 값이 주어지고 이는 실제 산업자동화 임베디드 기기에서 RLO의 역할을 수행한다. 그림 6에서 Tag1 = 1, Tag2 = 1, Tag3 = 0으로 가정하였을 때, and 노드에서 RLO = 1 비트와 연산을 수행하였을 때, 0의 결과를 얻는다. or의 경우 1의 결과를 얻는다.

해당 노드들의 결과값은 부모 노드의 매개 변수로 주어지며 or 결과는 1이 되며 최종적으로 %L20.0 메모리에 1의 결과값을 대입한다. %L은 외부 데이터 블록에서 참조된 데이터 값을 임시 저장하는 메모리 블록을 의미하며, %L20.0은 %L 메모리 블록의 20.0 주소를 의미한다. 해당 연산 순서는 산업자동화 제어 프로그램의 회로 연산 결과와 동일함을 확인할 수 있다. 본 연구에서는 비트로직 명령어를 모두 구현하여 산업자동화 제어 프로그램의 기본 요소인 직렬, 병렬처리, 접점, 코일의 구현이 완료되었다. 이후 추가적인 수식연산 등을 처리하는 S 社에서 제공하는 기본 함수에 대한 변환 규칙에 대해서 정의한다. 연구

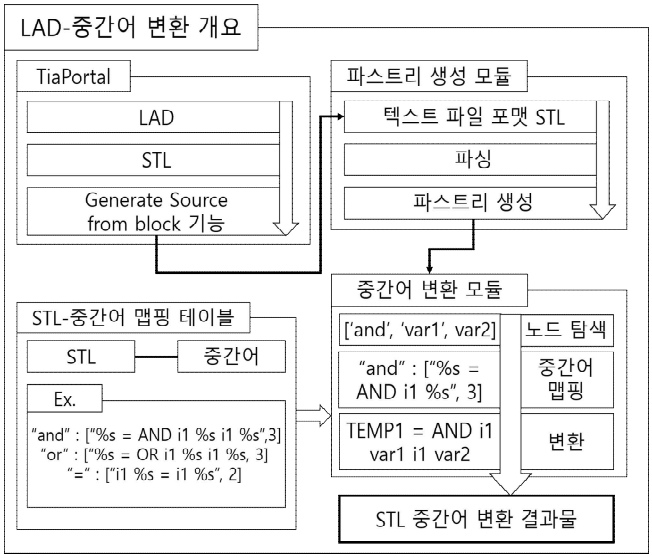

그림 5 중간어 변환 개요도

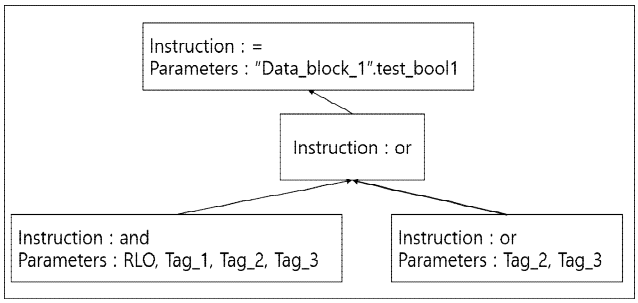

그림 6 파스트리 예시

대상인 S 社의 산업자동화 제어 프로그램의 명령어를 논리연산 명령어 집합, 수식연산 명령어 집합, 카운터 명령어 집합, 프로그램 제어 명령어 집합으로 분류하여 중간어를 구성한다. 비트로직 명령어는 회로의 접점 배치를 의미한다. 따라서 비트로직 명령어는 논리식 형태의 중간어로 변환하기 위하여 명령어의 의미 자체를 논리식에 적합하도록 치환하여야 한다. 비트로직 명령어는 파스트리 생성 시, 논리 연산을 의미하는 명령어로 치환한다. 예를 들어 산업자동화 제어 프로그램의 비트로직 라이브러리의 'A', 'O'는 and, or의 형태로 치환한다. 'A', 'O' 명령어는 각각 직렬, 병렬로 접점을 배치함을 의미하지만 치환된 형태의 'and', 'or' 명령어는 논리 연산을 의미한다. 표 5의 'Control Instruction'은 함수 호출, 주소 이동과 같은 프로그램 실행 흐름을 제어하는 명령어의 변환 규칙으로 구성된다. 'Math Instruction'은 수식 연산을 담당하는 명령어의 변환 규칙으로 구성된다. 'Counter Instruction'은 산업자동화 임베디드 기기의 카운터 관련 명령어의 변환 규칙으로 구성된다.

Smart Media Journal / Vol.7, No.1 / ISSN:2287-1322
http://dx.doi.org/10.30693/SMJ.2018.7.1.42

표 4 맵핑 테이블 예시

```
# 16-bit int math function
"+I" : "add i16",
"-I" : "add i16",
"*I" : "add i16",
"/I" : "add i16",
# 32-bit int math function
"+D" : "add i32",
"-D" : "sub i32",
"*D" : "mul i32",
"/D" : "div i32",
"MOD" : "urem i32",
# 32-bit float math function
"+R" : "add f32",
"-R" : "sub f32",
"*R" : "mul f32",
"/R" : "div f32",
# comparison function
"==D" : "icmp eq",
"<>D" : "icmp ne",
">=D" : "icmp sge",
"<=D" : "icmp sle",
">D" : "icmp sgt",
"<D" : "icmp slt"
```

4. 중간어 변환 프로그램 구현

중간어 변환 프로그램은 산업자동화 임베디드 기기 프로그램을 입력 받아 논리식으로 이루어진 파스트리 형태로 재구성하는 과정과 파스트리를 입력받아 중간어로 변환하는 과정을 수행한다. 프로그램은 Interface 모듈, Mapper 모듈, Converter 모듈로 구성된다. Interface 모듈은 중간어 변환 프로그램의 메인 루틴이다. Interface 모듈은 파스트리 형태로 변형된 산업자동화 임베디드 기기 프로그램을 입력받아 실제 산업자동화 임베디드 기기 프로그램 명령어 실행 순서에 따라, 맵핑을 진행한 후, 변환 결과값을 취합한다. Interface 모듈에는 실제 프로그램의 실행 과정을 반영하기 위하여 외부 데이터를 호출하는 Load Structure 모듈과 중간 연산값 저장 레지스터, 범용 레지스터가 추가적으로 구현되어 있다. Mapper 모듈은 입력 받은 노드의 명령어와 매개 변수 정보를 활용하여 소속된 라이브러리를 결정한 후, Converter 모듈의 대응되는 라이브러리 함수로 보낸다. Converter 모듈은 실제 각 명령어와 매개 변수에 대한 중간어 변환 기능을 포함한다. Converter 모듈에 포함된 'Math Instruction Converter', 'Counter Instruction Converter', 'Control Instruction Converter'는 각 라이브러리에 포함된 명령어들을 중간어로 변환하는 규칙이 정의되어 있다. Converter 모듈에는 새로운 라이브러리의 명령어를 추가하여 새로운 명령어에 대한 변환에 대응할 수 있다. 본 연구에서는 산업자동화 임베디드 기기의 레지스터와 메모리 구조를 활용한 실제 동작 방식을 구현하기 위한 모듈을 구현하였다. Load Structure 모듈은 산업자동화 제어 프로그램이 외부 데이터 블록의 데이터 값을 참조하는 과정을 구현한다.

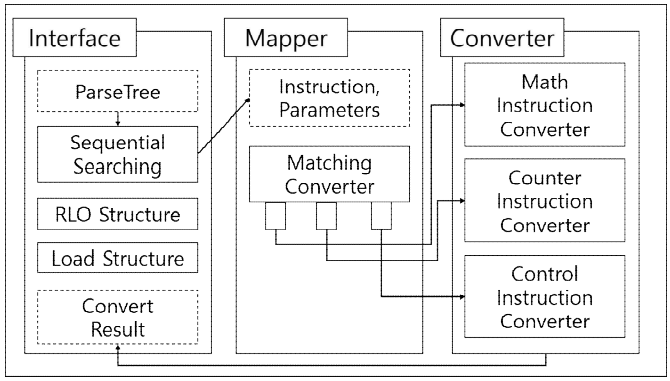

그림 7 중간어 변환 프로그램 개요도

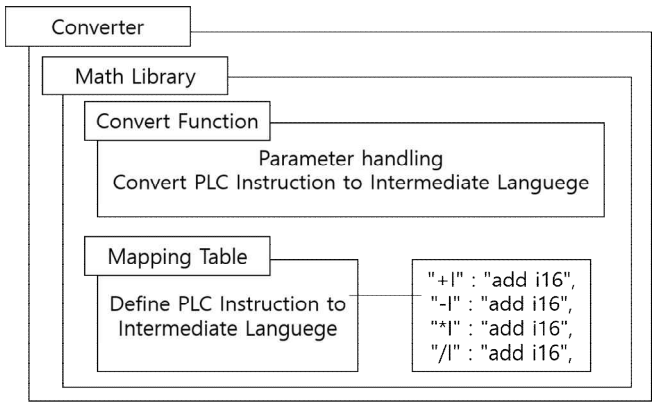

그림 8 Converter 모듈 개요도

산업자동화 임베디드 기기에서는 데이터 주소를 매개 변수로 가지는 L 명령어가 실행될 경우, Load 스택 프레임의 스택에 해당 값을 저장한다. 이후 해당 값을 활용하는 명령어의 매개 변수로 적재된 데이터 값을 할당시켜 주는 역할을 수행한다. 중간 연산값 저장 모듈은 프로그램의 접점의 개폐 여부를 확인하는 RLO의 데이터 값을 저장하기 위하여 구현되었다. 본 논문에서는 각 접점에서의 RLO의 데이터 값을 중간 연산값으로 정의한다. 따라서 초기 RLO = 1 이후의 RLO 값은 중간 연산값 저장 모듈에 저장된다. 중간 연산값 저장 모듈은 배열 형태로 구현되어 있으며, 각 배열의 공간은 저장값과 해당 공간의 할당/비할당 여부를 의미하는 flag 값을 포함한다. flag = 0일 경우 해당 공간은 할당이 가능함을 의미한다. flag = 1일 경우 공간에 이미 값이 할당되어 있어 할당이 불가능함을 의미한다. 중간 연산값이 매개 변수로 참조 될 경우, 해당 공간의 값은 초기화되며 flag는 0으로 초기화된다.

5. 변환결과

중간어 변환 프로그램을 활용하여 실제 변환을 진행하는 과정은 다음과 같다. 그림 6은 그림 3의 제어 프로그램을 논리식 형태로 변환한 파스트리이다. 파스트리 형태로 변환된 결과를 입력 받은 Converter 모듈은 다음과 같은 방식으로 변환을 진행한다. 그림 6에서 노드는 [Instruction : and, Parameters : RLO, Tag_1, Tag_2, Tag_3], [Instruction : or, Parameters : Tag_2, Tag_3], [Instruction : or], [Instruction : =, Parameters : “Data_block_1”.test_bool1] 순으로 탐색된다. 순서에 따라 탐색, 변환을 진행한 결과물은 표 6과 같다.

```
%TEMP1 = AND i1 RLO, “Tag_1”
%TEMP2 = AND i1 %TEMP1, “Tag_2”
%TEMP3 = AND i1 %TEMP2, “Tag_3”
%TEMP4 = OR i1 “Tag_2”, “Tag_3”
%TEMP1 = OR i1 %TEMP3, %TEMP4
Tag_3 = %TEMP1
```

표 5 산업자동화 제어 프로그램 중간어 변환 결과

첫 번째 탐색되는 노드는 %TEMP1 = AND i1 RLO, “Tag_1“, %TEMP2 = AND i1 TEMP1, “Tag_2“, %TEMP3 = AND i1 TEMP2, “Tag_3“로 변환된다. 두 번째 탐색되는 노드는 %TEMP4 = OR i1 ”Tag_2“의 연산 결과값이 세 번째 노드의 매개 변수로 입력된다. 따라서 세 번째 노드는 %TEMP1 = OR i1 %TEMP3, %TEMP4로 변환된다. 최종적으로 마지막 대입 연산자를 포함하는 노드는 Tag_3 = %TEMP1으로 변환된다.

6. 중간어를 활용한 산업 자동화 제어 프로그램 취약점 통합 분석 시스템

본 연구에서 제안한 산업자동화 제어 프로그램 중간어 변환 기술은 산업 자동화 제어 프로그램의 취약점 통합분석에 활용 가능하다. 산업 자동화 제어 프로그램에 대한 공격은 주로 데이터 및 프로그램 변조를 통하여 이루어진다. 따라서 프로그램 변조와 특정 데이터 변조로 인하여 시스템이 오작동 하는 공격 시나리오에 대한 대응이 필요하다. 따라서 본 연구에서는 중간어 기반 취약점 통합 시스템의 분석 도구로서 다음 분석 기법을 제안하였다. 첫 번째, 데이터 변조에 의한 시스템 오작동 가능성을 탐지하기 위해 기호실행 기반 퍼징 기술을 제안한다. 두 번째, 프로그램 변조를 탐지하기 위하여 프로그램에 대한 해쉬값을 저장하는 데이터베이스를 활용한 탐지 시스템을 제안한다. 그림 9는 중간어 변환 기술 기반 취약점 통합 분석 시스템의 예상 개요도이다.

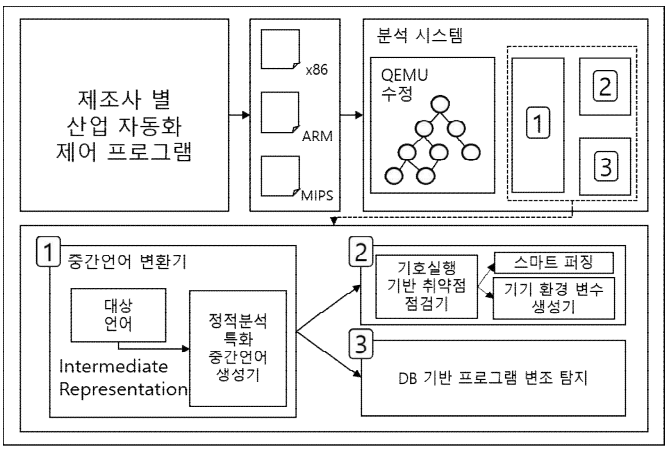

그림 9 산업 자동화 제어 프로그램 취약점 통합 분석 예상 개요도

III. 결 론

본 논문에서는 산업자동화 임베디드 기기의 산업자동화 제어 프로그램을 중간어 형태로 변환하는 기술을 제안하고, 실제 S 社의 산업자동화 제어 프로그램을 대상으로 하여 회로, 접점, 코일을 논리식 형태의 중간어로 변환하였다. 해당 기술을 활용함으로서 각 제조사 별로 상이한 문법과 명령어를 가진 산업자동화 임베디드 기기 프로그램에 대한 통합적인 관리와 분석을 진행할 수 있다. 또한 회로구조로 이루어진 산업자동화 제어 프로그램을 논리식으로 이루어진 중간어로 변환함으로서 프로그램에 대한 정량적 분석을 가능하게 할 수 있을 것으로 예상된다. 일반적인 프로그래밍 언어와 달리 그래픽 기반 언어인 산업자동화 제어 프로그램을 LLVM-IR 형태의 중간어로 변환한 결과는 다양한 형태의 프로그램 언어에 대하여 LLVM-IR로의 분석이 가능함을 시사한다. 그러나 현재 버전의 중간어 변환 프로그램은 S 社의 산업자동화 제어 프로그램 명령어에 대한 코드 커버리지를 향상 시킬 필요가 있다. 또한 본 연구의 결과물은 중간어로 변환함으로서 산업자동화 임베디드 기기에서 동작이 불가능하기 때문에 정적분석에 한정된다는 한계점을 가지고 있다. 따라서 향후 연구로서 코드 커버리지 향상과 LLVM의 프론트 엔드와 실제 동작을 담당하는 백 엔드를 연결하는 미들 엔드를 구현하여 실제 동작 가능한 시뮬레이션 환경을 구축하여 동적 분석에 대한 요구를 충족시킬 필요가 있다. 이를 위하여 S 社의 명령어 매뉴얼에 대한 조사를 필요로 한다. 추가적으로 동적분석 기능 추가를 위한 레지스터, 메모리 구조, LLVM 미들 엔드 외에도 산업 자동화 임베디드 기기의 취약점에서 높은 중요도를 가지는 네트워크 프로토콜 및 정보 송수신 기능에 대하여 분석이 필요하다.

Smart Media Journal / Vol.7, No.1 / ISSN:2287-1322
http://dx.doi.org/10.30693/SMJ.2018.7.1.44

REFERENCES

[1] Kushner, David. "The real story of stuxnet." ieee Spectrum 50.3, pp. 48-53, 2013.
[2] Spenneberg, Ralf, Maik Brüggemann, and Hendrik Schwartke. "PLC-blaster: A worm living solely in the PLC." Black Hat Asia, Marina Bay Sands, Singapore, 2016.
[3] Lattner, Chris, and Vikram Adve. "LLVM: A compilation framework for lifelong program analysis & transformation." Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization. IEEE Computer Society, 2004.
[4] Prähofer, Herbert, Florian Angerer and Rudolf Ramler. "Static code analysis of IEC 61131-3 programs: Comprehensive tool support and experiences from large-scale industrial application." *IEEE Transactions on Industrial Informatics*, vol. 13, no. 1 , pp. 37-47, 2017.
[5] Sandaruwan, G. P. H., P. S. Ranaweera, and Vladimir A. Oleshchuk. "PLC security and critical infrastructure protection." Industrial and Information Systems (ICIIS), 2013 8th IEEE International Conference on. IEEE, 2013.
[6] Lampérière-Couffin, Sandrine, O. Rossil,J.-M Roussel and J.-J Lesage, "Formal validation of PLC programs: a survey." Control Conference (ECC), 1999 European. IEEE, 1999.
[7] Alam, Shahid, R. Nigel Horspool, and Issa Traore. "MAIL: Malware Analysis Intermediate Language: a step towards automating and optimizing malware detection." Proceedings of the 6th International Conference on Security of Information and Networks. ACM, 2013.
[8] Kumar, R. Sravan, and Ashutosh Saxena. "Data integrity proofs in cloud storage." Communication Systems and Networks (COMSNETS), 2011 Third International Conference on. IEEE, 2011.
[9] *SIEMENS. SIMATIC Statement List(STL) for S7-300 and S7-400 Programming. reference Manual*

저 자 소 개

정병호(학생회원)

2018년 한양대학교 컴퓨터공학부 소프트웨어전공 학사 졸업.

<주관심분야 : 악성코드 탐지, 시스템 보안, 인공지능>

김용혁(학생회원)

2016년 한양대학교 컴퓨터공학부 소프트웨어전공 학사 졸업.
2000년 한양대학교 컴퓨터소프트웨어학과 석사 졸업.

<주관심분야 : 악성코드 탐지, SCADA 시스템>

배성일(학생회원)

2018년 한양대학교 컴퓨터공학부 소프트웨어전공 학사 졸업.

<주관심분야 : 악성코드 탐지, 유무선 네트워크 보안>

임을규(비회원)

1992년 서울대학교 컴퓨터공학 학사 졸업.
1994년 서울대학교 대학원 컴퓨터공학 석사 졸업.
2002년 서던캘리포니아대학교 컴퓨터공학 박사 졸업.

<주관심분야 : 악성코드 탐지, 유무선 네트워크 보안>