Journal of The Korea Society of Computer and Information
Vol. 22 No. 3, pp. 81-88, March 2017
www.ksci.re.kr
https://doi.org/10.9708/jksci.2017.22.03.081

Classification of Diagnostic Information and Analysis Methods for Weaknesses in C/C++ Programs

Kyungsook Han*, Damho Lee**, Changwoo Pyo***

Abstract

In this paper, we classified the weaknesses of C/C++ programs listed in CWE based on the diagnostic information produced at each stage of program compilation. Our classification identifies which stages should be responsible for analyzing the weaknesses. We also present algorithmic frameworks for detecting typical weaknesses belonging to the classes to demonstrate validness of our scheme. For the weaknesses that cannot be analyzed by using the diagnostic information, we separated them as a group that are often detectable by the analyses that simulate program execution, for instance, symbolic execution and abstract interpretation. We expect that classification of weaknesses, and diagnostic information accordingly, would contribute to systematic development of static analyzers that minimizes false positives and negatives.

▸ Keyword :Security, Weakness, Static Analysis, Diagnostic Information, Analysis Method

I. Introduction

전자정부 시스템 개발 사업을 중심으로 한 정부 지원 개발 사업에서 시작되어 최근 개발보안에 대한 요구가 확대되고 있는 상황이다. 이러한 환경 변화에 따라 소프트웨어 개발자 측면에서는 시큐어 코딩 규칙을 준수하여 개발하여야 한다는 인식이 확산되었고, 감리 측면에서는 이를 검사하기 위한 도구 사용이 확대되었다.

이에 따라 소스코드에 보안약점이 존재하는지 여부를 검사하기 위한 정적분석 도구에 대한 연구가 활발하게 진행되고 있다. 소프트웨어에서 나타나는 보안 취약점이 지속적으로 변화하고 있고 그에 따라 검사 대상 보안약점도 변화하고 있으므로, 정적분석 도구도 이에 따라 지속적으로 변화하여야 한다. 따라서 보안약점에 대하여 진단 방법을 기술하여 추가하는 방법을 제공하는 도구들이 존재하나, 이를 기술하기 위해서는 진단 방법 명세를 위한 표현 방법과 더불어 보안약점과 진단 방법에 대한 지식이 필요하다.

본 연구는 C/C++ 컴파일러의 각 단계에서 보안약점 분석을 위하여 어떤 작업이 추가적으로 필요한지에 대한 연구의 일환으로 진행되었다. CWE[1]에서 C/C++ 관련 보안약점을 분석한 정보를 바탕으로 보안약점 진단에 필요한 정보를 분류하고, 그 분류에 따라 컴파일러의 각 단계에서 수행할 수 있는 진단 방법에 대하여 연구하였다. 또한, SAMATE[2]에서 제공하는 정적 분석도구 벤치마크인 Juliet 테스트 모음[3]에서 검사하는 항목에 대하여 조사하였다. 또한 정적 분석도구에서 진단 규칙을 확장하기 위하여 사용하는 명세 언어에 대하여 연구하였다. 지티원의 SecurityPrism[4]에서 지원하는 규칙 명세 언어(RDL, Rule Description Language)[5][6]와 HP의 Fortify SCA (Static Code Analyaer)[7]에서 지원하는 진단 규칙 기술 방법에 대하여 분석하였다. 규칙을 기술하기 위한 언어는 보안약점 진단을 위한 정보를 표현하므로, 이러한 언어가 포함하는 정보에 대한 분석을 통하여 보안약점 진단에 사용하는 정보

• First Author: Kyungsook Han, Corresponding Author: Changwoo Pyo
*Kyungsook Han(khan@kpu.ac.kr) Dept. of Computer Engineering, Korea Polytechnic University
**Damho Lee(damho1104@gmail.com) Dept. of Computer Engineering, Hongik University
***Changwoo Pyo(pyo@hongik.ac.kr) Dept. of Computer Engineering, Hongik University
• Received: 2017. 03. 09, Revised: 2017. 03. 14, Accepted: 2017. 03. 23.
• This work was supported by the core technology R&D project of the Agency for Defense Development of Korea (UD160013ED)

와 진단 방법을 유추할 수 있다.

보안약점은 간단한 구문 요소를 검사함으로써 분석할 수 있는 단순한 형태부터 값의 흐름이나 범위를 분석해야 하는 경우에 이르기까지 다양하다. 구문 요소를 분석하는 방법은 단순한 구문구조 형식으로 표현이 가능하지만, 데이터나 제어의 흐름 정보가 필요한 경우에는 이러한 흐름에 대하여 표현할 수 있는 방법이 필요하다. 또한, 값이나 타입 정보, 사용자의 의도나 최적화 등 구문이나 제어/데이터 흐름으로 표현하기에 어려운 정보를 분석하여야 하는 보안약점의 경우가 있다. 보안약점의 분류는 보안약점을 진단하기 위하여 기술해야 하는 정보가 어떤 것인지 분석하기 위한 중요한 자료가 된다.

이러한 정보에 따라 소스 프로그램을 분석하는 과정에서 추출하여야 하는 데이터의 종류와 저장방식을 결정할 수 있다. 본 논문에서는 필요한 정보를 구문 정보, 흐름 정보, 타입 정보, 값 추적 정보로 구분하여 분석하였다. 분석 대상에서 '구문'은 문법정보 수준으로, 어휘 분석기나 구문 분석기 수준의 정보를 이용하여 검사가 가능하다는 것을 의미한다. '흐름'은 제어나 데이터 흐름 정보를 필요로 한다는 의미이다. '타입' 정보는 의미 분석기 정도의 수준에서 구할 수 있는 타입 정보를 의미하며, 이에 따라 값의 범위나 연산자를 제한하여 적용할 수 있다. '값 추적' 정보는 관련 데이터의 값에 대한 분석이 필요하다는 의미로, 많은 정적 분석도구에서 이러한 보안약점은 전용 분석기를 사용하여 분석하고 있다. 이러한 분류에 따라 진단 방법을 기술할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 다른 정적 분석도구에서 사용하는 규칙 기술 방법을 통해 어떤 정보를 표현하는지 간략히 소개한다. 3장에서는 진단 알고리즘을 표현하기 위한 정보의 표현 방법을 설명하고, 4장에서는 이 정보를 사용하여 의사 코드 방식으로 표현한 진단 규칙 기술 방법을 설명한다.

II. Related Works

보안약점 진단 방법에 대한 연구는 주로 정적 분석도구 개발과 관련하여 이루어지고 있다. 따라서 본 논문에서는 지티원의 SecurityPrism에서 지원하는 규칙 명세 언어와 HP의 Fortify SCA에서 지원하는 진단 규칙 기술 방법에 대하여 분석하였다. 또한, 정적 분석도구를 평가하기 위한 벤치마크로 사용되는 Juliet 코드에 대하여 분석하였다.

2.1 Rule Description Language (RDL)

지티원에서 제공하는 RDL은 단순 패턴과 흐름 패턴, 전용 분석기로 구분하여 진단을 위한 정보를 제공한다. 규칙 기술 언어는 BNF 형태로 정의하였으며[6], 이를 통하여 정적 분석도구를 확장하기 위한 규칙을 기술할 수 있다. RDL을 이용하여 명세를 기술하는 규칙은 크게 구문 패턴과 흐름 패턴으로 구분되며, 실제 도구에서는 범용 분석엔진에 의해 처리된다[5]. 구문 패턴은 특정 단어나 구문적 모양, 위치를 표현할 수 있고 흐름 패턴은 프로그램 상의 제어와 자료의 흐름을 표현한다.

이 방법은 기본적으로 보안약점을 검출하기 위하여 패턴으로 표현 가능한 정보를 표현하기 위한 체계를 잘 갖추어 놓은 것으로 보인다. 구문 패턴과 흐름 패턴으로 문법적 요소와 흐름 정보를 표현할 수 있도록 했고, 제어 흐름과 데이터 흐름의 여러 가지 분석 방법에 대해서도 표현 가능하다. 패턴으로 표현할 수 없는 보안약점에 대해서는 전용 분석기를 사용하도록 하고, 그 또한 유사한 방식으로 명시할 수 있도록 하였다.

본 연구에서 제안하는 구문 정보와 흐름 정보가 표현하는 바가 이와 유사하다고 할 수 있으나, 본 논문에서 제안하는 방식은 컴파일러 단계에 따라 타입 정보나 값 추적 정보에 대한 내용을 표현할 수 있고, 언어 특성이나 키워드에 종속되지 않는 방식을 연구한다는 점에서 차이가 있다.

2.2 Custom Rules of Fortify SCA

Fortify SCA에서 사용하는 규칙 기술 방법은 XML을 사용하여 규칙을 기술한다. 프로그램의 속성을 모델링하여 규칙화할 수 있는 구조를 가지고 있으며, 직관적으로 요소나 속성을 표현하도록 되어있다. 또한, 여러 개의 Rule들을 통합하여 복합적인 Rulepack으로 구성할 수 있고, 동일한 Rule도 버전을 나누어 관리가 가능하다. 기본적인 Common Rule을 기반으로 제어흐름, 자료흐름, 구조적, 컨텐츠 및 환경의 네 가지 분석 목적에 따라 요소나 속성을 추가하거나 수정하여 작성할 수 있도록 제공하고 있다.

이 방법은 XML 표현 구조를 이용하여 구조적으로 기술 가능하고 특정 부분에서 분석할 요소를 직관적으로 이해할 수 있다는 장점이 있다. 그러나 컴파일러 단계에 따라 추출할 수 있는 정보에 의해 진단 방법을 기술하는 본 논문에서의 접근 방법과는 차이가 있다. 또한, 본 논문에서는 컴파일러 진행 단계에 따라 어떤 정보를 얻을 수 있는지, 즉, 각 단계에서 추출할 수 있는 진단 정보가 무엇인지를 표현함으로써 진단 방법 개발에 자연스럽게 컴파일러 기술을 접목하도록 한다.

2.3 Juliet test suite

Juliet 테스트 모음[12]은 미국 국가안보국의 CAS (Center for Assured Software)에서 개발되었다. 미국 국립표준기술연구소에서 정적 분석기의 성능을 평가하기 위한 시험코드로 줄리엣 테스트 모음을 채택했다. Juliet 테스트 모음은 CWE 항목별로 분류되어 있으며, 하나의 CWE 항목 내에서도 테스트 코드를 유형별로 분류하여 제공한다. 따라서 정적 분석기의 테스트에 있어 유형 별로 검출 성능을 평가할 수 있으며, 다른 정적 분석기와 객관적인 성능 비교를 가능하게 한다. Juliet 테스트 모음은 독립성을 가지며 CWE를 기반으로 JAVA와 C/C++로 나뉘어 있다. 테스트 하고자 하는 보안약점 항목이 있는 경우,

해당 CWE에 해당하는 제목으로 테스트 코드를 찾아 사용할 수 있다.

본 논문에서는 C/C++언어와 관련된 보안약점 항목을 추출하고, 그에 관련된 Juliet 코드를 분석하였다. 85개 보안약점을 대상으로 관련 43개 항목의 테스트 코드를 분석하였으며, 이를 통하여 각 보안약점 진단을 위해 필요한 정보를 분류하였다.

III. Representation of Diagnostic Algorithms

본 논문에서는 보안약점에 대한 진단 알고리즘을 표현하기 위해서 간단한 명세언어와 같은 형태를 사용하였다. 추상 구문 트리(AST, Abstract Syntax Tree)[8][9]와 제어 흐름 그래프(CFG, Control Flow Graph)[8][9], 데이터 흐름 그래프(DFG, Data Flow Graph)[8][9]가 존재한다고 가정하고 보안약점 진단에 필요한 정보를 기술하였다. 또한 유사한 특성을 가지는 함수나 이름의 그룹이 지정되어 있다고 가정하고 그룹 명세를 사용하였으며, 값 추적에 의한 분석을 위하여 값의 범위를 표현할 수 있도록 하였다. 본 논문에서 사용하는 명세 언어는 완성된 형태의 언어는 아니며, 보안약점의 진단방법을 표현하기 위한 도구로 사용되었다고 볼 수 있다.

3.1 Syntax Representation

구문 표현은 어휘(lexical) 수준에서 검사할 수 있는 정보와 구문 트리(syntax tree) 구조가 필요한 정보를 표현하기 위한 방법을 의미한다. 스캐너 수준의 정보 표현을 위해 정규표현식을 사용할 수 있어야 하며, 구문 구조를 위해서는 구문 트리에 해당하는 정보를 표현하기 위한 방법이 필요하다. 이를 위해 식별자와 같은 이름, 변수나 파라미터를 나타낼 수 있는 기호, 수식과 문장의 형태, 제한요건 등의 정보를 표현할 수 있어야 한다. 또한 구문 트리의 자식 노드를 통한 에지나 그래프에서의 관계 표현처럼 포함이나 연결과 같은 관계를 표현하는 방법을 제공하여야 한다.

이를 위해서는 추상 구문 트리 정보를 표현해주는 것으로 가능하며, 일반적으로 추상 구문 트리의 노드와 자식노드 관계, 속성 등의 정보를 이용하여 표현한다. 예를 들어 트리 노드를 문장이나 함수 호출, 연산자 사용과 같이 분류하기 위하여 노드의 타입이라는 의미로 nType 요소를 사용하였다. 이 외에도 자료형을 표현하기 위한 type, 함수 이름이나 변수 이름과 같은 문자열로 표현되는 정보인 name, 자식 노드를 표현하는 child와 같이 추상 구문 트리에서 사용하는 속성들을 표현하였다. 이 외에도 상수 값을 직접 노드에 표현할 수 있거나 추정 가능한 경우 사용할 수 있도록 value 속성도 사용하였다. 앞에서 언급한 것과 같이 완성된 형태의 언어는 아니므로 노드 구성을 위한 구조체를 표현하지는 않았다. 또한 필요에 따라 속성 정보를 추가할 수 있도록 하고 노드의 구분을 위하여 이름을 명시할 수 있도록 표현하였다. 노드 정보와 관련하여 사용한 속성 명칭은 표 1.과 같다.

Table 1. Node Attributes of Abstract Syntax Tree

Name	Meaning
nType	Information for node classification. Names such as ASSIGN, CALL
type	The type of node
name	Information expressed as a string, such as a function or variable name
child	Child node in abstract syntax tree
value	If the value can be expressed directly, the value

보안약점을 나타내는 구문 정보를 검사하기 위해서 “match ~ where ~” 형태의 키워드를 사용하였다. match 절에서는 노드 종류에 따라 해당되는 노드를 찾아 식별자를 대응시킨다. 이 때 조건을 기술하기 위하여 where 절을 사용하였다. 이 표현은 일종의 질의어 형태로 볼 수 있으며, 추상 구문 트리에서 조건에 부합하는 노드를 찾아 이름을 부여하고, 조건이 부합되면 보안약점으로 반환한다.

3.2 Flow Information Representation

흐름 정보 표현을 위해서는 제어의 흐름과 데이터 흐름을 구분하여 표현할 수 있도록 해야 한다. 데이터 흐름의 경우, 복사와 같이 데이터가 직접 전달되는 경우와 수식에서 포함되어 영향을 미치는 간접 흐름을 구분하여 표현할 수 있으나, 본 논문에서는 흐름 그래프 상에서 에지의 개수를 제한할 수 있도록 하는 방법을 사용하였다.

흐름 정보는 제어 흐름 정보와 데이터 흐름 정보로 구분할 수 있다. 제어 흐름은 cf, 데이터 흐름은 df로 구분하며, 모든 경로에서의 흐름은 “_must”를 추가하여 표현한다. 규칙을 기술할 때 “-cf-”, “-df-”, “-cf_must-”, “-df_must-”와 같이 에지 형태로 표현하고 화살표를 이용하여 방향성을 표현한다. 간접 흐름과 직접 흐름을 구분하여 표현하지 않으므로, 에지의 개수에 제한이 필요한 경우 [a..b] 형태로 에지 표현 뒤에 표시할 수 있도록 하였다. 예를 들어, -df->[1..3]이라고 기술하는 경우 데이터 흐름 에지가 1개에서 3개 사이인 경우를 표현한 것이다. 이를 이용하면 직접 데이터 흐름인 경우, [1..1]로 표현하여 구분할 수 있다. 직접 데이터 흐름과 간접 데이터 흐름을 따로 구분하지 않는 이유는 제어 흐름과의 통일성을 유지하고 표현의 확장 가능성을 주기 위해서이다.

흐름 정보는 두 개 이상의 구문 정보와 그 사이의 흐름 정보 형태로 표현하였다. 두 개 이상의 노드를 식별하여 이름을 부여하고, 그 이름 사이의 흐름 정보를 표현하는 형태이다. 노드 식별을 위해서는 앞 절에서 설명한 구문 정보 검사 형태인 “match ~ where ~” 형태를 사용하였다. 구문 정보 사이의 흐름 정보는 if 문 형태를 사용하여 표현하였다.

3.3 Representation for Grouping

구문을 표현할 때, 여러 개의 이름을 그룹으로 묶어서 그룹에 속하는지 여부를 검사해야 하는 경우가 있다. 유사한 용도로 사용되는 여러 개의 함수 중 하나를 표현해야 하는 경우, 그 이름을 모두 명시하기보다는 함수 그룹을 먼저 지정하고 그 그룹에 속하는지 여부를 표현하는 방식을 사용하였다. 예를 들어, 메모리 할당을 위한 malloc, calloc, realloc, new와 같은 함수 중 하나의 함수를 표현하기 위하여 memAlloc이라는 그룹을 만들어 그 안에 포함되는지 검사할 수 있다. 그룹에 포함되는지 여부를 표현하기 위하여 'in'이라는 키워드를 사용하였다. 이러한 방법은 함수나 타입, 값의 범위 등을 그룹으로 지정함으로써 표현 방식에 확장성을 부여할 수 있다. 즉, 미리 정해진 이름이나 타입을 사용하기보다는 상황에 따라 그룹에 포함되는 요소를 변경함으로써 검사할 수 있는 대상을 변경할 수 있게 된다.

3.4 Other Additional Expressions

앞 절에서 설명한 표현 방법과 더불어 특정 기능을 수행하기 위하여 함수 호출 형태로 기술할 수 있도록 하였다. 이는 실제 함수를 구현하여 사용한다는 것을 의미하지는 않으며, 진단 방법 기술을 위하여 기능적 요소를 간단하게 표현하기 위한 방법으로 사용한 것이다. 또한 흐름 정보 표현에서 에지의 수를 제한한 것과 같이 [a..b]와 같은 형태를 사용하여 값의 범위를 표현할 수 있도록 하였다. 이는 값 추적 검사를 위하여 값의 범위를 제한하기 위한 용도로 사용할 수 있다.

IV. Weaknesses and Diagnostic Method

4.1 Diagnosis by Syntax Informations

구문 정보는 소스코드를 파싱하는 단계를 통하여 추출 가능한 정보를 의미한다. 어휘 정보는 스캐너 수준에서 분석 가능한 정보로, 어떤 함수 이름을 사용하였는가 여부를 진단하는 정도의 간단하게 표현할 수 있는 정보이다. 예를 들어, gets()와 같은 안전하지 않은 함수를 사용하였는지 여부를 검사한다면, 검사해야 할 문자열 정보로 표현할 수 있을 것이다. 안전하지 않은 함수 리스트를 각각 정규표현식으로 표현하여 그 합을 구하면 "취약한 API 사용(CWE-676)"에 해당하는 규칙을 기술할 수 있다.

구문 트리 정보는 어휘 정보보다는 복잡한 정보를 필요로 하나, 구문 트리 상에서 구문 정보로 파악 가능한 정보를 의미한다. 예를 들어, 구문 트리 상에서 switch 문장에 해당하는 노드의 하위 노드로 default 문이 포함되어 있는지 여부를 검사하는 규칙을 기술할 수 있을 것이다. 이 규칙을 이용하면 "default 케이스가 없는 switch 문(CWE-478)"에 해당하는 보안약점을 진단할 수 있다.

표 2.는 구문 정보를 이용하여 진단할 수 있는 보안약점의 예와 이를 분석하기 위한 알고리즘을 2절에서 설명한 표현 방법으로 기술한 것이다.

구문 정보에 대한 진단방법 표현은 2절에서 기술한 것과 같이 추상 구문 트리 상의 관계로 표현할 수 있다. 예를 들어, '포인터에 고정 주소를 저장하는 보안약점(CWE-587)'에 대한 진단방법 표현은 추상 구문 트리 상에서 대입 연산이 있는 노드를 찾는 것으로 시작된다. 이후 저장될 변수(LHS)의 타입이 포인터이고 저장되는 값(RHS)이 상수인지 여부를 검사하도록 기술한다. 이 때, 저장되는 값에 대해 kind라는 속성 항목을 사용하였으며, CONST라는 값을 가지는지 검사함으로써 상수 값 여부를 판단하도록 표현하였다. 또한 '디폴트 항목이 없는 switch 문(CWE-478)'의 경우에는 switch 문에 해당하는 노드를 먼저 검사한 후 마지막 항목에 default로 되어있지 않으면 보안약점으로 진단한다. 이 때 switch 문의 마지막 case 항목이 마지막 자식 노드에 저장되어 있는 구조를 표현하기 위하여 child(last)라는 표현을 사용하였다. case의 label에 해당하는 값은 name 속성 값을 이용하여 검사할 수 있도록 표기하였다.

4.2 Diagnosis by Flow Informations

흐름 정보는 제어 흐름 분석과 데이터 흐름 분석 단계에서 얻을 수 있는 정보로 표현할 수 있다. 소스 코드를 기반으로 분석하므로 제어 흐름 분석의 단위는 문장으로 볼 수 있으며, 기본적으로 함수 내에서의 흐름 분석을 대상으로 한다. 일반적으로 앞 절에서 설명한 구문 정보와 함께 사용된다.

제어 흐름 정보는 문장 사이의 실행 순서에 따른 정보를 의미한다. 데이터 흐름 정보는 제어 흐름 정보를 기반으로 데이터

Table 2. Diagnosis using Syntax Information

Weakness	Diagnostic Algorithm	Algorithm Representation
CWE-587: Assignment of a Fixed Address to a Pointer	1) Check the assign operation on the AST 2) Check whether the pointer type is used at the 1'st child (LHS) node 3) Check whether 2'nd child (RHS) node contains a constant value	match (n.nType == ASSIGN) where (n.child(0).type == pointer) and (n.child(1).kind == CONST) return n
CWE-478: Missing Default Case in Switch Statement	1) Check the switch statement on the AST 2) Check the label of last child node 3) Check whether the label is not "default"	match (n.nType == SWITCH) where(n.child(last).name != "default") return n

Table 3. Diagnosis using Flow Information

Weakness	Diagnostic Algorithm	Algorithm Representation
CWE-415: Double Free	1) Check that function call named free 2) Check the first parameter of that function 3) Check whether there is a free function call for the same variable in the reverse direction of the data flow	match (n.nType == CALL) where (n.name == “free”) match (m.nType == CALL) where (m.name == “free) match (s.nType == CALL) where (s.name in memAlloc) if (match (m –cf-> n) and (match (s –df->[1..1] m) and (match (s –df_must->[1..1] n) where (m.param(0) == n.param(0))) return n
CWE-762: Mismatched Memory Management Routines	1) Check that function call named free or delete 2) Check the first parameter of that function 3) Checking for the same variable in the reverse direction of the data flow 4) Check that the memory allocation function used for variable definition matches	match (n.nType == CALL) where (n.name in memFree) match (m.nType == CALL) where (m.name in memAlloc) if (match e:(m –df_must->[1..1] n) where (m.child(0) == n.param(0)) and mismatch(m.name. n.name)) return n

가 직접 또는 간접적으로 영향을 미치는 관계를 의미한다. 흐름 정보를 필요로 하는 보안약점은 제어와 데이터의 흐름에 의하여 어떤 데이터가 영향을 미치는지 여부에 의해 진단할 수 있는 보안약점이다.

예를 들어, 메모리 할당 후 해제하는 과정에서 new()를 이용해 객체를 생성한 후 free()로 해제하려 하거나 malloc()으로 할당한 메모리를 delete() 하려 하는 등 호환되지 않는 루틴을 사용한다면 “일치하지 않는 메모리 관련 루틴 사용(CWE-762)” 보안약점으로 진단할 수 있다. 이 보안약점은 free() 함수 호출로의 제어 흐름상에 malloc()이 도미네이터(dominator)[8][9]로 존재하는지 여부를 검사하는 방식으로 진단 가능하다. 표 3.은 흐름 정보를 이용하여 진단할 수 있는 보안약점의 예를 보여주고 있다.

‘중복된 free 보안약점(CWE-415)’의 경우 동일한 변수에 대한 두 개의 free() 함수 호출이 동일한 변수에 대해 이루어지고, 동일한 메모리 할당 문장에서 두 free() 문으로의 데이터 흐름이 존재하면 보안약점으로 진단하는 것을 표현한 것이다. 메모리 할당에서 free()로의 데이터 흐름은 변수가 그대로 전달되는 직접 흐름이므로 [1..1]을 사용하였다. 또한, 메모리 할당에서 해제로의 데이터 흐름은 반드시 있어야 하므로 df_must를 사용하였다.

‘메모리 관리 함수의 부적절한 연결(CWE-762)’의 경우 메모리 관리 함수가 잘 못 연결되었을 때 생기는 보안약점으로, malloc()으로 할당한 메모리를 delete()로 해제하거나 new()로 할당한 메모리를 free()로 해제할 때 발생한다. 진단방법 표현에서 memFree와 memAlloc은 함수의 그룹을 표현하는 것으로, memFree는 free()와 delete()와 같은 메모리 해제 함수를, memAlloc은 malloc(), calloc(), new()와 같은 메모리 할당 함수를 그룹화 한 것이다. mismatch() 또한 두 개의 함수 이름 사이에 연결 관계를 따로 설정하였다고 가정하고 사용하였다. 이 경우, 메모리 할당과 해제 사이에 반드시 데이터 흐름이 존재해야 하므로 df_must를 사용하였다.

이러한 그룹 사용이나 함수 형태의 표현은 진단방법 표현 방법에 확장성을 주기 위한 것으로, 기본적인 구문 트리나 제어 흐름, 데이터 흐름 그래프를 가정하고 다른 요소에 대해서는 변경이나 확장이 가능하도록 하기 위하여 사용한 방법이다.

4.3 Diagnosis by Type Informations

타입 정보로 표현되는 보안약점은 의미 분석 단계 중에서 얻어지는 타입 정보를 사용하여 분석할 수 있는 보안약점을 의미한다.

단순한 구문 정보에 타입 정보를 추가해야 분석 가능한 보안약점의 예를 보면, ‘숫자 잘림 오류(CWE-197)’과 같은 보안약점을 들 수 있다. 대입문에 대한 구문 트리를 분석할 때 저장될 값을 계산하는 수식 부분의 타입과 값이 저장될 변수의 타입을 비교함으로써 이러한 보안약점이 발생할 가능성이 있는지 검사할 수 있다. 또한 형 변환의 경우에도 변환 대상 값과 변환되는 타입 사이에서 동일한 문제가 발생할 수 있다. 이를 분석하기 위해서는 구문 구조뿐 아니라 각 노드에 해당하는 타입 정보를 필요로 하게 된다.

타입 정보는 구문 구조의 각 요소에 타입 정보를 추가하도록 구조화 할 수 있다. 타입 정보는 컴파일 과정에서 심볼 테이블 정보를 활용하여 구문 트리에 정보를 추가하여 검사 가능하며, 특정 타입에 대한 허용 여부를 기술하기 위한 요소를 추가하면 분석한 타입 정보와 비교하여 보안약점 진단에 활용할 수 있다.

표 4.에서는 타입 정보를 사용하여 보안약점을 진단하는 예를 보여준다. ‘숫자 잘림 오류(CWE-197)’ 보안약점을 진단하기 위해서는 추상 구문 트리에서 대입문이나 형 변환에 대한

Table 4. Diagnosis using Type Information

보안약점	분석 방법	진단방법 표현
CWE-197: Numeric Truncation Error	* For Assignment statement 1) Check the assign operation on the AST 2) Check whether the data type of LHS is integer type or real type 3) Check whether the data type of RHS is integer type or real type 4) If both are true, check the range of the LHS and RHS. 5) Check whether the range of the LHS is narrower. * For Type Casting 1) Check the cast operation on the AST 2) Check whether cast type is integer type or real type 3) Check whether the data type of operand is integer type or real type 4) If both are true, check the range of the cast type and the range of the operand. 5) Check whether the range of the cast type is narrower.	match (n.nType == ASSIGN) or (n.nType == CAST) where (n.child(0).type in numeric) and (n.child(1).type in numeric) and !wide(n.child(0).type, n.child(1).type) return n

노드를 먼저 검사한다. 두 경우 모두 숫자에 해당하는 자료형인지 여부를 먼저 검사하고, 대입될 값 또는 형 변환될 값의 자료형과 저장될 변수 또는 형 변환 대상 자료형을 비교한다. 숫자에 해당하는 자료형은 numeric이라는 그룹으로 만들고, 그룹에의 포함 여부를 확인한다. 이 때 wide라는 함수 형태로 표현한 것은 첫 번째 파라메터가 두 번째 파라메터보다 값의 영역이 넓은 것을 의미하는 것이다. 이는 두 파라메터 모두 숫자에 해당하는 자료형인 경우에만 사용 가능하며, 실제로 존재하는 함수가 아닌 의미를 전달하기 위한 함수 형태의 표현이다.

4.4 Diagnosis using Value Analysis Information

값 추적 분석은 보안약점 여부를 진단하기 위하여 어떤 지점에서 변수 또는 수식이 가질 수 있는 값의 범위를 추정하는 것이 필요한 경우를 의미한다. 값에 대한 분석은 의미 분석 과정이나 최적화 과정에서 상수 값에 대한 일부 정보를 얻을 수 있으나, 입력 값과 같이 실행 과정에서 얻을 수 있는 정보를 필요로 하므로 동적 분석이 필요한 영역으로 판단할 수 있다.

이에 해당하는 보안약점의 예를 들면, '경계를 벗어난 읽기(CWE-125)'나 '힙 영역에서의 버퍼 오버플로우(CWE-122)'와 같은 것이 있다. 첨자의 사용 범위가 지정된 버퍼 영역을 벗어나는 값을 가지는지 여부를 판단해야 하는 문제이므로, 버퍼의 정의나 사용 부분에서 변수가 사용되는 경우 값의 범위를 추론하여야 한다. 첨자에 해당하는 변수가 외부에서 입력되었거나 연산에 의해 계산되는 경우, 첨자 값의 범위를 분석하여 실제 문제가 발생할 수 있는지 여부를 점검해 볼 수 있을 것이다. 외부에서 입력된 값이 연산과 분기문을 통해 전달되는 경우, 값의 범위 분석은 복잡한 연산이 필요한 문제가 된다.

값의 범위를 분석하기 위한 방법으로는 동적 분석 방법과 기호 기반 실행(symbolic execution)[11], 요약 해석(abstract interpretation)[12] 기법을 사용한다. 동적 분석 방법은 실행 과정에서 얻을 수 있는 정보를 이용한 분석이므로 컴파일 단계에 따라 보안약점을 분류하는 본 논문의 영역에서 벗어난다. 기호 기반 실행 기법은 프로그램에서 사용하는 변수를 구체적인 값이 아닌 심볼(symbol)로 표현하고 분기문을 만나면 기호에 대한 제약식을 추가하면서 모든 가능한 경로에 대해 분석하는 방식으로 값의 범위를 추론하는 방법이다. 경로가 통합되는 경우에도 각각의 범위를 유지하며 값을 추론하기 때문에 경로 폭발로 인한 시간과 메모리 소모가 심하다는 문제가 있다. 따라서

Table 5. Diagnosis using Value Information

보안약점	분석 방법	진단방법 표현

CWE-122: Heap-based Buffer Overflow	1) Checks whether a memory allocation function is called
	2) If memory allocation size is constant, set the size to max value. Set the value to 0 if the variable can not be evaluated
	3) Check the variable usage according to data flow information
	4) If the index is a constant, check whether the value is between 0 and max
	5) If the index is a variable, check whether the range of the value is between 0 and max. If the range is unknown, set the index to infinite.
	6) Check if the value is out of range at 4) or 5)
		match (n.nType == CALL)
		where (n.name in memAlloc)
		if (n.name == "malloc")
		max = n.child(1),value
		else if (n.name == "calloc")
		max = n.child(1).value * n.child(2).value
		else if (n.name == "realloc")
		max = n.child(2).value
		match (n -df_must->[1..1] m)
		if (n.child(0).name == m.child(x).name)
		and !m.child(x).index in [0..max]
		return m

분기문과 반복문이 많이 사용되는 경우 너무 많은 분기로 인해 부정확한 분석이 되는 경우가 발생한다.

요약 해석 기법은 실제 값을 요약 값으로 사상하고 프로그램을 요약 영역 (abstract domain) 상에서 해석하는 방식이다. 추상 영역의 값을 사용하여 실제 실행과 가까운 정보를 얻을 수 있으므로 동적 분석을 실행하는 것과 유사한 효과를 기대할 수 있다. 예를 들어, {1, 3, 4, 7}의 값에 대해 '1에서 7 사이의 정수'로 요약하는 방식이다. 프로그램에서 변수가 가지는 값의 범위와 연산, 실행 경로 등을 요약함으로써 프로그램 전체 내용을 추론하며, 분기에서는 나누어서 값을 추론하나 경로가 합쳐지는 경우에는 값의 범위 역시 통합하여 요약하게 된다.

값 추적 분석과 관련하여 표현 가능한 정보는 허용 가능한 값의 범위를 기술할 수 있도록 요소를 추가한 것이다. 진단을 위한 규칙으로 값의 범위를 지정하기 위해서는 변수나 상수에 의한 값의 범위를 분석하는 기법이 사용되어야 하므로 실제 값의 범위를 적용하는 것은 매우 복잡한 문제가 된다. 허용 범위를 기술함에 있어서 상수만 허용할 것인지, 기호 기반 실행이나 요약 해석 기법을 고려하여 변수를 사용한 범위를 허용할 것인지에 따라 실제 구현에서 분석이 가능한 범주와 엔진의 복잡도가 달라질 것이다.

표 5.는 값 추적 정보를 사용하여 보안약점을 진단하는 예를 보여준다. '힙 영역에서의 버퍼 오버플로우(CWE-122)' 보안약점의 경우, 동적으로 할당된 메모리에 대하여 그 영역을 벗어난 사용을 하는지 검사하여야 한다. 이를 분석하기 위하여 메모리 할당에 대한 함수 호출 부분을 먼저 검사한다. 함수에 따라 할당 받는 메모리의 크기를 계산할 수 있는 방법이 다르므로 그 값에 대한 계산 부분은 파라메터의 값을 이용하여 계산하는 형태로 기술하였다. 이 때 메모리 할당 함수 호출 시 주소 값이 저장되는 변수를 첫 번째 자식노드로, 파라메터들을 그 이후의 자식 노드로 가진다고 가정하여 child 속성의 첨자를 사용하였다. 할당된 메모리를 사용하는 부분에 대해서는 직접 데이터 흐름이 존재할 것이므로 -df_must->[1..1]로 데이터 흐름을 표현하였다. n.child(0).name은 주소 값을 가지는 변수 이름이며, 변수 사용과 관련된 노드 m에 대하여 동일한 이름을 가지는 자식 노드를 표현하기 위하여 child(x)라는 기호를 첨자에 사용하였다. 이 노드에 대하여 첨자를 index라는 속성으로 표현하여, 값의 범위에 포함되지 않으면 보안약점을 검출하도록 기술하였다.

V. Conclusions

보안약점을 분석하기 위하여 필요한 정보를 분류하고 이러한 정보를 컴파일러 각 단계에서 얻을 수 있는 정보와 연계하여 기술하였다. 어휘 분석과 구문 분석, 의미 분석 단계에서 분석 가능한 보안약점과 추가적인 분석 정보를 필요로 하는 보안약점에 대하여 분류하였다. 또한, 각 분류에 해당하는 보안약점에 대한 진단 방법을 간단하게 정의한 표현 방법을 사용하여 기술하였다. 이를 통하여 컴파일러 단계에서 정적 분석을 위하여 취득하여 활용할 수 있는 정보에 대하여 정리하였다.

본 논문은 컴파일러 단계에서 정적 분석을 위하여 취득할 수 있는 정보와 그 활용에 대한 연구의 일환이다. 모든 보안약점이 이러한 정보로 분석 가능한 것은 아니며, 각 보안약점에 대하여 전용 분석 방법을 사용해야 하는 경우가 존재한다. 그러나 이러한 보안약점 분류와 분석 정보 기술 방법을 통하여 정적 분석 도구의 확장을 위하여 필요한 정보를 기술할 수 있는 토대가 마련될 것으로 기대한다. 향후 분석 방법을 기술하는 언어에 대하여 좀 더 완성도를 높여 나가야 할 것이다.

REFERENCES

[1] CWE, Common Weakness Enumeration, *http://cwe.mitre.org/*

[2] NIST, *https://samate.nist.gov/*

[3] SAMATE, Juliet Test Suite v1.2 for C/C++ User Guide, National Security Agency

[4] SecurityPrism, *http://www.gtone.co.kr/kr/security-static-analysis-tools.php*

[5] Hyun-Joon Kwon, Hyunha Kim, Kyung-Goo Doh, "Developing An Automatic Tool for Static Detection of Software Security Vulnerabilities", pp.37-40, KIISE, Vol. 28.2, Feburary 2010 (in Korean)

[6] Hyunha Kim, Tae-Hyoung Choi, Seung-Cheol Jung, Oukseh Lee, Kyung-Goo Doh, Soo-Yong Lee, "Rule-based Source-code Analysis for Detection of Security Vulnerability", WISA2009:The 10th International Workshop on Information Security Applications, Busan, South Korea, August 25~27, 2009

[7] Fortify Static Code Analyzer, *https://saas.hpe.com/en-us/software/sca*

[8] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, "*Compilers: Principled, Techniques, and Tools*", Addison Wesley, 1986

[9] Steven S. Muchnick, "*Advanced Compiler Design and Implementation*", Morgan Kaufmann, pp.169-265, 1997

[10] C. Cadar, and K. Sen, "Symbolic execution for software testing: three decades later," Communications of the ACM, 56.2, pp.82-90, July 2013.

[11] P. Cousot, and R. Cousot, "Abstract interpretation: A

unified lattice model for static analysis of programs by construction or approximation of fixpoints," Proceedings of the 4th ACM SIGACT- SIGPLAN symposium on Principles of programming languages, pp.238-252, ACM, January 1977.

Authors

Kyungsook Han received the B.S., M.S. and Ph.D. degrees in Computer Engineering from Hongik University, Korea, in 1993, 1995 and 2002, respectively

Dr. Han joined the faculty of the Department of Computer Engineering at Korea Polytechnic University, Gyeonggi-Do, Korea, in 2003. She is currently an Associate Professor in the Department of Computer Engineering, Korea Polytechnic University. She is interested in compiler optimization and software security,

Damho Lee received the B.S. degree in Computer Engineering from Korea Polytechnic University, in 2013 and the M.S. degree in Computer Engineering from Hongik University, in 2016, respectively

Damho Lee is currently a Ph.D student in the Department of Computer Engineering, Hongik University. He is interested in software security, program optimization.

Changwoo Pyo received the B.S. degree in Electronic Engineering and the M.S. degree in Computer Engineering from Seoul National University in 1980 and 1982. He received the Ph.D. degree in Computer Science from University of Illinois at Urbana-Champaign in 1989. Dr. Pyo joined the faculty of the Department of Computer Engineering at Hongik University, Seoul, Korea, in 1991. He is currently a Professor in the Department of Computer Engineering, Hongik University. He is interested in programming language, software security, program optimization, and parallel computing.