보안공학연구논문지
JSE
Journal of Security Engineering

SECURITY ENGINEERING RESEARCH
SUPPORT CENTER

보안공학연구논문지

Journal of Security Engineering

ISSN : 1738-7531 제 14권,제 1호,2017년2월

목 차

보안공학연구논문지 *Journal of Security Engineering*
Vol.14, No.1 (2017), pp.21-32
http://dx.doi.org/10.14257/jse.2017.02.04

추상구문트리 메타모델을 통한 코드의 보안 취약성 가시화

손현승[1], 김영철[2]

A Visualization of Secure Vulnerability in Code using Abstract Syntax Tree Metamodel

Hyun Seung Son[1], R. Young Chul Kim[2]

요 약

최근 정보 시스템들이 해킹되어 개인정보 유출 사고가 빈번하게 발생함에 따라 정부는 시큐어 코딩 가이드를 제정하고 전자정부 관련 정보화 사업에 의무적으로 적용하도록 하였다. 그러나 일반개발자가 시큐어 코딩의 모든 소프트웨어의 취약점을 알고 구현하기란 쉽지 않다. 그리고 기 개발된 소프트웨어에 취약점이 있는지 확인하기는 어렵다. 그러므로 소스코드 레벨에서 보안 취약성을 자동으로 분석하고 그 결과를 가시화하는 기술이 필요하다. 본 논문에서는 소프트웨어의 가시화 기술을 시큐어 코딩에 적용하여 보다 쉽게 보안 취약성을 분석할 수 있는 방법을 제안한다.

핵심어 : 메타모델, 보안 취약점, 시큐어 코딩, 추상구문트리

Abstract

Recently, as information systems have been hacked and personal information leakage accidents have occurred frequently, the korea government enacted a secure coding guide and made it mandatory for informatization projects related to the electronic government. However, it is not easy for general developers to know and implement all software vulnerabilities in secure coding. Also, it is difficult to check for vulnerabilities in previously developed software. Therefore, there is a need for techniques to automatically analyze security vulnerabilities at the source code level and visualize the results. In this paper, we propose a method to analyze security vulnerability more easily by applying software visualization technology to secure coding.

Keywords : metamodel, secure vulnerability, secure coding, abstract syntax tree

Received(January 03, 2017), Review request(January 04, 2017), Review Result(1st: January 19, 2017, 2nd: January 31, 2017)

Accepted(February 06, 2017), Published(February 28, 2017)

[1]Dept. Computer Information Comm., Hongik Univ., Sejong-ro, Jochiwon-eup, Sejong, 30016, Korea
email: son@selab.hongik.ac.kr
[2](Corresponding Author) Dept. Computer Information Comm., Hongik Univ., Sejong-ro, Jochiwon-eup, Sejong, 30016, Korea
email: bob@hongik.ac.kr

* 이 논문은 2015년 교육부와 한국연구재단의 지역혁신창의인력양성사업의 지원을 받아 수행된 연구임 (NRF-2015H1C1A1035548)

ISSN: 1738-7531 JSE

1. 서론

최근 모 인터넷 쇼핑업체가 사이버 범죄자에 의해 해킹당해 1,000만 건의 개인정보가 유출되었고 회사를 상대로 개인정보 유출 협박해 금품을 요구하는 사고가 있었다. 이때 사용된 해킹 기술이 APT(Advanced Persistent Threat) 공격이다[1]. 이 APT 공격은 취약점을 이용해 악성코드를 공격 대상자 몰래 설치하고 실행하게 하는 것을 말한다.

이렇게 소프트웨어 취약점으로 인해 각종 정보유출 피해가 과거부터 지금까지 끊임없이 일어남에 따라 행정안전부에서는 시큐어 코딩 가이드를 제공하고 이를 의무화시키고 있다[2]. 이것은 앞 사례와 같이 실제 해킹의 75%가 소프트웨어 취약점을 악용해 이루어지고 있기 때문이다[3]. 또한 운영단계에서의 취약점 제거 비용이 개발단계보다 60~100배의 비용 소요(IBM보고서)가되므로 개발단계에 적용되는 시큐어 코딩은 비용절감 효과도 있다[4].

행정안전부의 시큐어 코딩 가이드는 국외의 소프트웨어 약점 및 취약성을 분류하고 정리하는 단체인 CWE(Common Weakness Enumeration)를 참조해 만들어졌다[2][5]. CWE의 보안 취약성은 프로그램 언어 및 운영체제와 같은 환경 등에 따라 약 1,000개 이상으로 구성되어 있다. 그러나 개발자가 시큐어 코딩을 위해서는 모든 소프트웨어의 취약점을 알고 이를 대처 가능하도록 소프트웨어 구현을 해야 한다. 일반적인 개발자들은 이런 취약점 항목에 대한 전문적인 지식 습득이 어렵고 취약점을 어떻게 수정해야하는지 잘 모르는 경우가 많다. 그러므로 소스코드 레벨에서 취약점 자동으로 분석해주는 도구가 필요하다.

정보통신산업진흥원(NIPA)의 소프트웨어 가시화(SW Visualization)는 소프트웨어의 비가시성을 극복하고 개발 과정의 투명성을 확보를 통해 품질 확보 및 품질 문제의 조기 검출해 비용 절감할 수 있는 서비스이다[6]. 우리는 NIPA의 SW Visualization 시즌 I, II, III과정을 통해 기존 가시화 방법을 추상구문트리 메타모델(Abstract Syntax Tree Metamodel, ASTM)[7]을 활용한 시스템으로 확장하였고 다양한 가시화 방법들(응집도, 결합도, 재사용성, 성능, 저전력 등)을 연구 중이다[8-10].

본 논문에서는 기존의 가시화 방법을 시큐어 코딩에 적용하기 위해서 추상구문트리 메타모델을 사용해 코드 보안취약점을 찾아낼 수 있는 방법을 제안한다. 제안한 방법은 xCodeParser[11]를 이용해 추상구문트리 메타모델을 자동 생성하고 생성된 구문트리에서 보안 코드 규칙과 탐색알고리즘을 사용해 보안 위협이 되는 요소를 식별하고 이를 가시화한다. 추상구문트리 메타모델을 사용하기 때문에 다양한 프로그래밍 언어에 같은 방법을 적용가능하고 기존 도구에서 발견하지 못하는 보안 위협 등을 발견할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구를 소개한다. 3장에서는 보안취약성을 자동으로 발견할 수 있는 가시화 방법에 대해 설명한다. 4장에서는 제안한 방법을 적용사례를 보여주고, 5장에서는 결론 및 향후 연구에 대해서 언급한다.

2. 관련연구

2.1 행정안전부의 시큐어 코딩 가이드

시큐어코딩 가이드는 소프트웨어 개발과정 중 코드 작성시 발생할 수 있는 개발자 실수, 논리적 오류 등으로 인한 보안취약점을 최소화하기 위한 방법이다. 국내에서는 2009년부터 전자정부서비스 개발에서 시큐어 코딩 관련 연구를 진행하였고, 2012년까지 전자정부지원사업 등을 대상으로 SW 보안약점 시범진단을 수행하였다. 이러한 SW 보안약점 제거 · 조치 성과에 따라 2012년에 행정안전부 '정보시스템 구축 · 운영 지침'이 개정 · 고시됨으로써 전자정부 관련 정보화 사업 수행 시 적용이 의무화 되었다[2].

이 가이드는 정보시스템 구축 시 가장 많이 사용되는 개발언어인 Java와 C언어의 코딩 가이드를 제공하고 있다. 코딩 가이드는 보안약점설명과 보안대책이해를 위한 코딩 예제(Bad/Good)를 제시한다. [표 1]과 같이 보안약점의 유형은 입력데이터 검증 및 표현, 보안기능, 시간 및 상태, 에러 처리, 코드 오류, 캡슐화, API 오용으로 7가지로 나누어진다.

[표 1] 시큐어코딩 가이드의 보안약점의 유형
[Table 1] The types of security weaknesses in secure coding guide

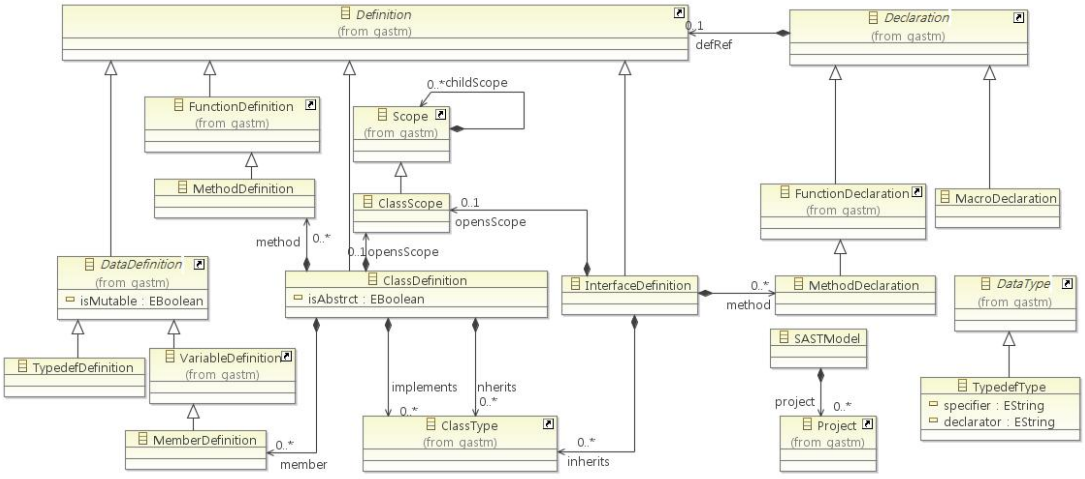
유형	설명
입력데이터 검증 및 표현	프로그램 입력 값에 대한 검증 누락 또는 부적절한 검증, 데이터의 잘못된 형식지정으로 인해 발생할 수 있는 보안약점
보안기능	인증, 접근제어, 기밀성, 암호화, 권한관리 등을 적절하지 않게 구현하면 발생할 수 있는 보안약점
시간 및 상태	동시 또는 거의 동시 수행을 지원하는 병렬 시스템 환경에서 시간 및 상태를 부적절하게 관리하여 발생할 수 있는 보안약점
에러 처리	에러를 처리하지 않거나, 불충분하게 처리하여 에러정보에 중요정보가 포함될 때 발생할 수 있는 보안약점
코드 오류	변환 오류, 자원의 부적절한 반환 등과 같이 개발자가 범할 수 있는 코딩오류로 인해 유발되는 보안약점
캡슐화	중요한 데이터 또는 기능성을 불충분하게 캡슐화하였을 때, 인가되지 않는 사용자에게 데이터 누출이 가능해지는 보안약점
API 오용	의도된 사용에 반하는 방법으로 API를 사용하거나, 보안에 취약한 API를 사용하여 발생할 수 있는 보안약점

시큐어 코딩 가이드를 통해 보안약점 및 대응기법 등 전반에 대한 이해가 가능하다. 또한, 개발자는 소프트웨어 개발 시 참조할 수 있고 발주자는 요구사항 도출 시 보안약점을 진단 제거에 참조할 수 있다.

2.2 추상구문트리 메타모델

OMG(Object Management Group) 표준인 추상구문트리 메타모델(ASTM)는 컴파일러나 파서에서 사용되는 추상구문트리(AST)의 메타모델이다[7]. 이 ASTM의 목표는 소프트웨어 개발, 소프트웨어 현대화를 위한 도구, 플랫폼, 이종 분산 환경의 메타데이터 저장소 사이에서 자세한 소프트웨어 구조를 쉽게 교환하기 위한 것이다. 특히, ASTM은 다른 판매회사의 다양한 도구를 공유할 수 있는 추상구문트리를 대표하는 모델링 요소에 대한 규격이 정의 되어있다. 또한, ASTM은 C, C++, C#, Java, Ada, VB/.Net, COBOL, FORTRAN, Jovial 등과 같이 많은 수의 프로그래밍 언어들을 표현 할 수 있다.

그러나 OMG의 ASTM은 구현되어 있지 않고 메타모델의 규격만 정의되어 있다. 그리고 그 메타모델의 구조는 복잡하게 만들어져 있다. 우리는 기존의 ASTM의 메타모델을 확장하여 [그림 1]과 같이 정의 하였다. 그 확장한 ASTM은 표준 문서의 193개의 메타모델 구조들을 구현하였고 소프트웨어 가시화를 위해 필요한 10개의 구조물들을 추가하였다[11].



[그림 1] 확장된 추상구문트리 메타모델 설계

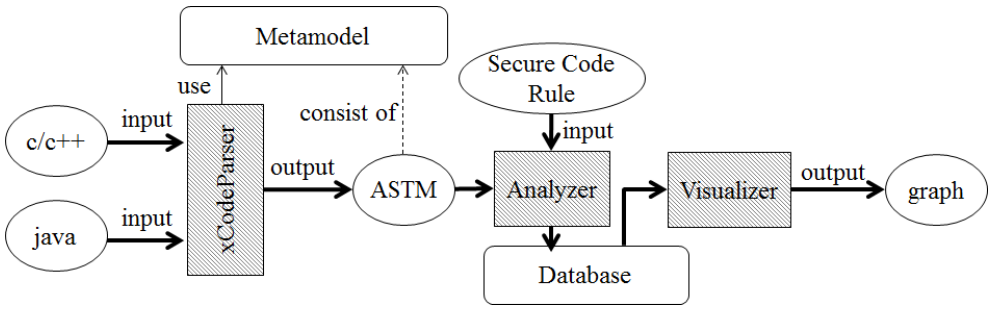
[Fig. 1] A design of the extended Abstract Syntax Tree Metamodel (ASTM)

xCodeParser는 코드 분석기를 거쳐 프로그래밍 언어의 입력 코드로 부터 ASTM 파일을 생성할 수 있다. XMI(XML Metadata Interchange)[12]을 기반으로 하는 생성된 ASTM 파일은 MDA(Model Driven Development) 기반의 기존 도구들과 상호 운영할 수 있다. 이 ASTM은 클래스, 변수, 함수, 구문, 조건 등 프로그램언어의 모든 요소를 트리로 표현된다.

3. 보안 취약성 가시화 방법

3.1 보안 소프트웨어 가시화 체계

소프트웨어 가시화 도구를 개발하기 위해서는 기본적으로 파서와 가시화 도구가 필요하다. 본 논문에서는 기존 가시화 구조[13]에서 보안 위배 코드의 가시화를 위한 방법으로 [그림 2]와 같이 보안 코드 분석기를 추가하여 확장한다. 이 가시화 방법은 xCodeParser, 분석기, 가시화의 3가지 단계로 구성되어 있다.



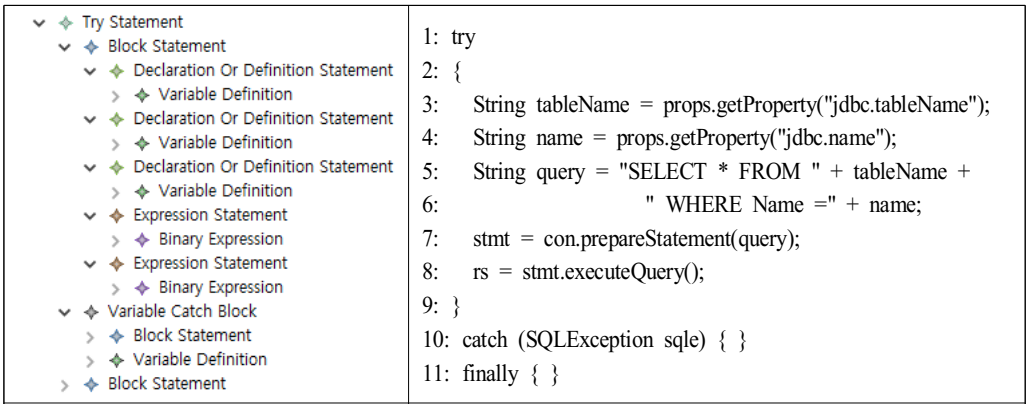
[그림 2] 보안 위배 코드의 가시화 방법

[Fig. 2] The method of security violation codes' visualization

xCodeParser는 C, C++, Java와 같은 프로그래밍 언어로부터 ASTM을 생성한다. 분석기는 생성된 ASTM을 읽어 코드의 보안 취약성을 찾기 위해서 보안 코드 규칙을 입력으로 각각의 보안 위배사항을 찾는다. 이 결과는 데이터베이스에 저장되고 가시화 도구를 통해서 그래프로 출력한다.

3.2 보안 위험 코드의 추적 및 가시화

보안 코드의 추적 및 가시화를 위해서는 보안에 문제가 있는 코드의 유형을 파악하고 추상화하여야 한다. [그림 3]은 안전하지 않은 코드의 예를 나타낸 것으로 입력 데이터 검증 및 표현 파트의 SQL 삽입에 관한 예제이다. SQL 삽입은 공격자가 시스템의 입력란에 SQL을 삽입하여 데이터베이스로부터 정보를 열람하거나 조작하는 보안약점이다. 안전한 코딩을 위해서는 preparedStatement 클래스와 함께 사용하기를 권장한다. 하지만 그림의 코드에서는 preparedStatement를 사용하였지만 문자열에 다른 변수 값을 허용하여 보안에 문제가 생기는 예이다. 이와 같은 경우 단순히 코드 내에 preparedStatement와 같은 API 사용여부로 찾을 경우 보안 위배사항을 검색하지 못한다.

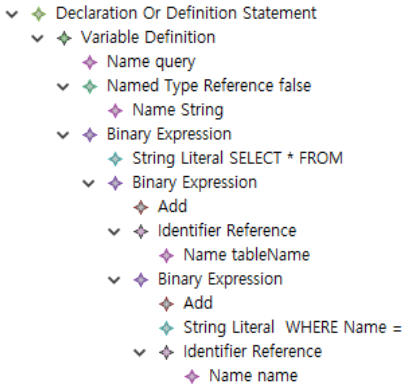


(가) ASTM의 구조 (나) 안전하지 않은 코드의 예(Java)

[그림 3] SQL 삽입 공격 위험 코드 예

[Fig. 3] The example of risk code from SQL injection attack

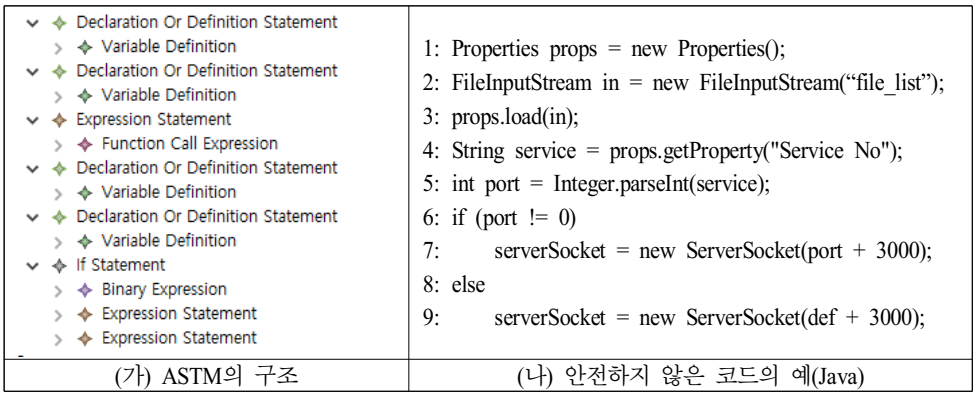
[그림 3]의 5~6라인의 ASTM의 구조는 [그림 4]와 같다. 변수 query는 String 타입으로 하위 노드로 Binary Expression을 갖는다. 이 Binary Expression은 오퍼레이터 Add를 중심으로 LHS는 문자열 “SELECT * FROM”, RHS는 하위 Binary Expression을 갖는 구조로 반복된다. 그러므로 이 경우의 코딩 규칙을 찾기 위해서는 먼저 SQL 구문의 문자열을 찾아야 하고 상위에서 정의된 변수가 이 SQL 구문의 문자열 안에서 변수명이 사용될 경우를 찾아야 한다.



[그림 4] SQL 삽입 관련 코드의 ASTM

[Fig. 4] The ASTM of code related to SQL injection

자원삽입은 외부 입력 값을 검증하지 않고 시스템 자원을 그대로 사용하는 경우를 말한다. 이 경우 공격자는 입력 값 조작을 통해 임의의 접근, 수정, 충돌을 유발할 수 있다. 이때 사용되는 자원은 파일, 명령어 파라미터 값 등이 해당한다. [그림 5]는 자원 삽입 공격 위험 코드의 예와 이에 대한 코드를 ASTM을 표현한 것이다.

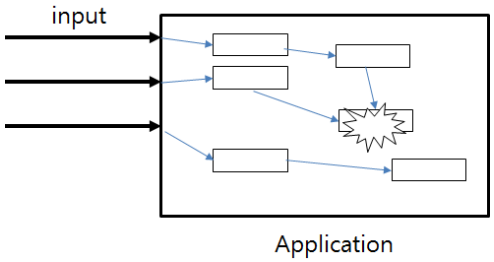


(가) ASTM의 구조 (나) 안전하지 않은 코드의 예(Java)

[그림 5] 자원삽입 공격 위험 코드 예

[Fig. 5] The example of risk code from resource injection attack

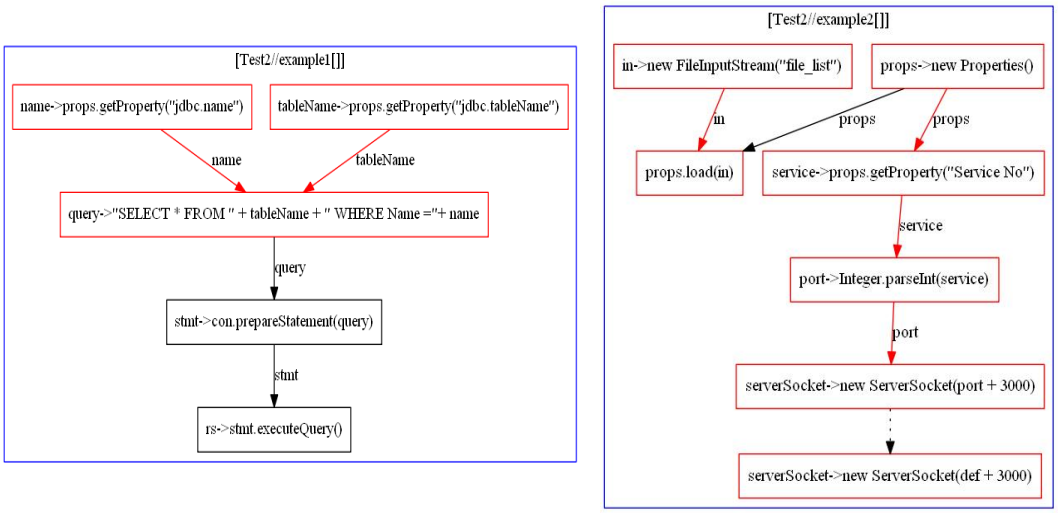
이러한 보안 위협 코드들의 살펴본 결과 [그림 6]과 같이 프로그램의 입력될 수 있는 파라미터, 파일, 명령어, 사용자인터페이스를 경유하여 공격이 시도된다. 그러므로 입력 값 확인 등과 같은 보안 코드를 구현하지 않았을 시 경우에 따라서 치명적인 보안약점으로 사용될 수 있다. 본 논문에서는 코드 상의 보안 문제점을 추적하고 가시적으로 확인 할 수 있도록 코드 문장의 연결 관계를 그래프로 표현한다.



[그림 6] 소프트웨어 보안 취약점의 공격 경로

[Fig. 6] The attack path of software vulnerability

[그림 7]은 [그림 3]과 [그림 5]의 코드 문장과의 관계를 가시적으로 표현한 것이다. 이 가시화는 코드의 문장과의 관계를 노드와 엣지로 표현한다. 노드에는 [그림 8]과 같이 코드의 좌항에 변수이름이 있을 경우 그 이름을 표현해주고 "->" 기호와 함께 코드의 우항을 표시한다. 변수이름이 없을 경우는 코드의 문장을 그대로 표현한다. 그리고 제어문과 반복문은 노드로 표현되지 않고 엣지에 나타난다.

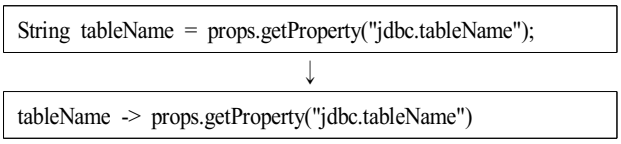


(가) SQL 삽입 예 (나) 리소스 삽입 예

[그림 7] 소프트웨어 보안 취약점의 가시화 결과

[Fig. 7] The visualization result of software vulnerability

엣지는 노드와 노드를 연결해주는 선으로 실선과 점선으로 구분된다. 실선은 코드의 현재 문장이 다음문장에 영향을 미치는 경우에 영향을 미치는 변수 이름과 같이 표현한다. 즉, 현재 문장에서 정의된 변수가 다음문장에서 사용된 경우를 말한다. 실선은 제어문에 의해 분기된 경우를 표현한다.



[그림 8] 코드 문장을 노드로 변환

[Fig. 8] The transformation to a node from a statement of code

이 코드 가시화 알고리즘은 [그림 9]와 같다. 먼저 코드를 입력으로 ASTM으로 변환한다. ASTM을 순차적으로 클래스 내의 모든 메서드를 탐색한다. 메서드 내부에서 Declaration Or Definition Statement를 만나면 변수의 이름은 좌항에 초기 값은 우항에 레이블로 표시하고 노드를 만든다. Expression Statement를 만나면 그 중에서 Function Call Expression은 메서드 이름이 Qualified Over Data일 경우(예) abc.method()), 레이블을 표시하고 노드를 만든다. 그리고 메서드의 파라미터인 Actual Parameter Expression의 모든 데이터를 검색해서 기존에 만들어진 노드가 있는지 검사해 노드를 만든다. 양변이 있는 Binary Expression중 Equal은 좌변과 우변의 값을 레이블에 표시하고 노드를 만든다. 나머지 if, while, for 와 같은 모든 statement는 expression만 조사에서 기

존 노드와 관계가 있는 표현한다. 모든 구문을 검사해 노드가 완성되면 노드의 이름을 매치해 그래프의 엣지를 만들어 낸다. 그래프가 완성되면 보안 코드 규칙을 참조하여 위배되는 코드를 적색으로 표시해 준다.

```
algorism GenerateGraph
    outASTM ← run xCodeParser with filename
    funDefes ← visit FunctionDefinition with outASTM
    call VisitStatement with funDefes
    for each funDef in funDefes do
        statements ← get BlockStatement in funDef
        for each statement in statements do
            call VisitStatement with statement
algorism VisitStatement is input: statement
    if statement = DeclarationOrDefinitionStatement then
        exp ← get Initializer in statement
        label ← exp to string
        call VisitExpression with exp
        call MakeNode with lable
        label ← get name in statement
        call MakeNode with lable
    else if statement = ExpressionStatement then
        call VisitExpression with exp ← get Expression in statement
    else if statement = ForStatement then
        call VisitExpression with exp ← get Initializer in statement
        call VisitExpression with exp ← get Updater in statement
        call VisitExpression with exp ← get Expression in statement
        call VisitStatement with statement ← get BlockStatement in statement
    else if statement = IfStatement or WhileStatement then
        call VisitExpression with exp ← get Expression in statement
        call VisitStatement with statement ← get BlockStatement in statement
algorism VisitExpression is input: exp
    if exp = FunctionCallExpression then
        label ← get Name in exp
        call MakeNode with label
        params ← get ActualParameterExpression in exp
        for each param in params do
            call VisitExpression with param
    else if exp = BinaryExpression then
        op ← get Operator in exp
        if op = Equal then
            call MakeNode with op ← get RightOperand in exp
            call MakeNode with op ← get LeftOperand in exp
```

[그림 9] 가시화를 위한 그래프 생성 알고리즘

[Fig. 9] The algorism of graph generation for visualization

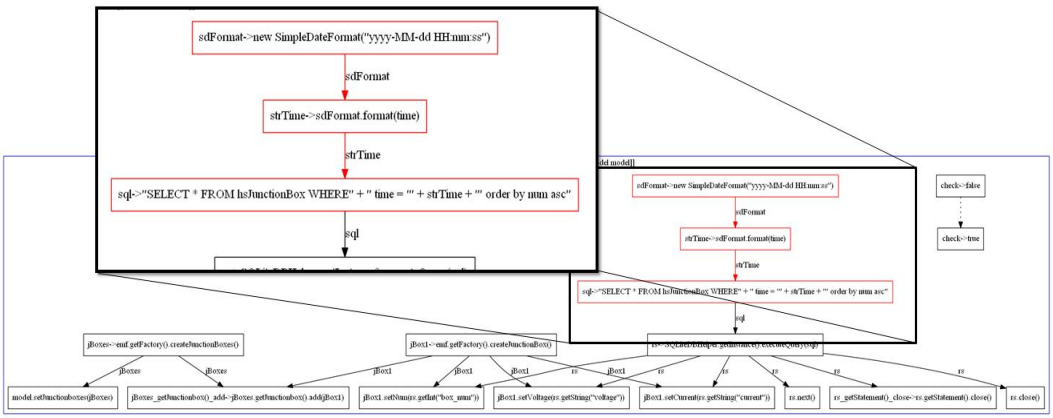
4. 적용사례

본 논문에서는 적용사례로 [그림 10]과 같이 HS솔라에너지의 태양광 모니터링 시스템(PVMS) [14]에 대해서 제안한 방법으로 보안성 체크를 수행해 본다. 태양광 모니터링 시스템은 태양광 셀로부터 수집된 전력의 양과 온도, 기울기 센서 들의 정보를 모니터링하고 관리하기 위한 시스템이다. PVMS는 접속함과 인버터에서 생산된 데이터를 RS232통신으로 태양광 발전이 설치된 지역에서 관리하는 모니터링 시스템인 로콜 서버에 전달한다. 이렇게 지역에 있는 로콜 서버들은 TCP/IP를 통해서 통합 모니터링 서버로 전달되고 모든 데이터는 통합되고 데이터베이스에 저장한다.

(가) 로콜 모니터링 (나) 통합 모니터링

[그림 10] 태양광 모니터링 시스템의 구조

[Fig. 10] The structure of Photovoltaic Monitoring System



[그림 11] 태양광 모니터링 시스템의 보안 취약점 가시화 결과

[Fig. 11] The visualization result of software vulnerability in Photovoltaic Monitoring System

태양광 모니터링 시스템은 데이터를 수집하기 위해서 TCP/IP 네트워크를 사용하기 때문에 보안에 취약할 수 있다. 그러므로 보안에 위배되는 코드들이 있는지 검사하기 위해서 각 클라이언트로부터 수집 받은 데이터를 데이터베이스에 저장하는 부분의 코드에 대해서 가시화를 수행해 보았다. 그 결과 [그림 11]과 같이 SQL 구문 사용한 부분 중 strTime이라는 시간을 가져오는 코드에서 SQL 삽입 위협이 있는 코드를 발견하였다.

5. 결론

소프트웨어의 규모가 커지면 커질수록 보안의 위협은 계속되고 있고 한번 사고가 나면 천문학적인 피해를 보게 된다. IBM보고서와 같이 운영단계에서의 취약점 제거 비용이 개발단계보다 60~100배의 비용 소요되므로 보안은 개발 완료 후 수정하는 것보다 개발 단계에서 예방하는 것이 효율적이다. 본 논문에서는 행정안전부의 시큐어 코딩 가이드를 활용하여 개발자가 쉽게 발견할 수 있도록 가시화 방법을 제안하였다. 제안한 방법은 개발자가 시큐어 코딩에 대한 전문적인 지식 습이 없더라도 활용할 수 있도록 소스코드 레벨에서 취약점 자동으로 분석해주는 도구를 개발하였다. 이 방법은 xCodeParser를 이용해 추상구문트리 메타모델을 자동 생성하고 생성된 구문트리에서 보안 코드 규칙과 탐색알고리즘을 사용해 보안 위협이 되는 요소를 식별하고 이를 가시화한다. 추상구문트리 메타모델을 사용하기 때문에 다양한 프로그래밍 언어에 같은 방법을 적용가능하고 기존 도구에서 발견하지 못하는 보안 위협 등을 발견할 수 있다.

하지만 현재는 하나의 메서드 내에서만 보안 위협 경로를 찾을 수 있다. 향후 연구로 이를 확장하여 메서드와 메서드, 모듈과 모듈 간까지 추적하여 광범위한 부분에 대해서 보안 위험을 확인할 수 있도록 연구 중이다. 또한 더 많은 시큐어 코딩 규칙을 적용 중이다.

References

[1] G. Thomson, APTs: a poorly understood challenge. Network Security. **(2011)**, Vol.2011, No.11, pp. 9-11.

[2] Ministry of Public Administration and Security (MOPAS), Java Secure Coding Guide for developer and operator of e-government software, MOPAS and Korea Internet & Security Agency **(2012)**.

[3] C. H. Paik, J. M. Sun, K. D. Ryu, B. J. Moon, T. W. Kim and W. J. Kim, Android-based mobile messenger application vulnerability analysis and secure coding method. The Korean Society Of Computer And Information. **(2014)**, Vol.22, No.1, pp.83-87.

[4] https://www.isixsigma.com/industries/software-it/defect-prevention-reducing-costs-and-enhancing-quality/, March 1 **(2009)**.

[5] S. C. Coley, J. E. Kenderdine, L. Piper and R. A. Martin, Common Weakness Enumeration (CWE) : A Community-Developed Dictionary of Software Weakness Types, Version 2.9, MITRE **(2015)**.

[6] S. E. Lee, S. H. Baek, S. J. Kang, W. M. Song, S. E. Lee, W. S. Jang, D. G. Jung and J. M. Cho, SW Visualization : Software Development Quality Management Manual, NIPA **(2013)**.

[7] http://www.omg.org/spec/ASTM/1.0/, January 1 **(2011)**.

[8] C. Y. Seo, S. Y. Moon, H. S. Son, G. S. Yi, Y. S. Kim and R. Y. C. Kim, Open Source of Integration Service Tools for Venture & Small Business Maintenance Solutions. Korea Information Processing Society Review. **(2016)**, Vol.23, No.6, pp. 22-32.

[9] G. H. Kang, R. Y. C. Kim, G. S. Yi, Y. S. Kim, Y. B. Park and H. S. Son, A Practical Study on Code Static Analysis through Open Source based Tool Chains. KIISE Transactions on Computing Practices. **(2015)**, Vol.21, No.2, pp. 148-153.

[10] B. K. Park, H. E. Kwon, H. S. Son, Y. S. Kim, S. E. Lee, R. Y. C. Kim, A Case Study on Improving SW Quality through Software Visualization, Journal of KIISE, Vol.41, No.11 935-942 **(2014)**.

[11] H. S. Son, Y. S. Kim, Y. B. Park, W. Y. Kim and R. Y. C. Kim, Abstract Syntax Tree Metamodel for SW Visualization. Proceedings of the 4th International Conference on Convergence Technology, **(2014)**, July 2-5; Manila, Philippines.

[12] http://www.omg.org/spec/XMI/2.1.1/, December 1 **(2007)**.

[13] H. S. Son, S. Y. Moon, R. Y. C. Kim and S. E. Lee, Replacing Source navigator with Abstract Syntax Tree Metamodel(ASTM) on the open source oriented tool chains SW Visualization. Proceedings of the 5th International Conference on Convergence Technology, **(2015)** June 29-July 2; Hokkaido, Japan.

[14] H. S. Son and R. Y. C. Kim, Modeling a Photovoltaic Monitoring System based on Maintenance Perspective for New & Renewable Energy. Proceedings of the 2nd International Joint Conference on Convergence, **(2016)** January 18-22; Hanoi, Vietnam.

