Journal of The Korea Society of Computer and Information
Vol. 20 No. 8, pp. 69-75, August 2015
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2015.20.8.069

Design and Implementation of Software Vulnerability Analysis Algorithm through Static Data Access Analysis

Hyun-il Lim*

Abstract

Nowadays, software plays various roles in applications in wide areas. However, the security problems caused by software vulnerabilities increase. So, it is necessary to improve software security and safety in software execution. In this paper, we propose an approach to improve the safety of software execution by managing information used in software through static data access analysis. The approach can detect the exposures of secure data in software execution by analyzing information property and flows through static data access analysis. In this paper, we implemented and experimented the proposed approach with a base language, and verify that the proposed approach can effectively detect the exposures of secure information. The proposed approach can be applied in several areas for improving software safety by analysing vulnerabilities from information flows in software execution.

▸ Keyword : static data access analysis, software vulnerability analysis, static software analysis

I. Introduction

실생활에서 컴퓨터의 활용이 일반화되고, 인터넷을 통한 다양한 정보 검색 및 공유, 소셜 네트워크와 같은 다양한 서비스가 활발히 이용되고 있다. 이런 환경에서 인터넷을 통해 방대한 양의 정보들이 전달되고 있으며, 다양한 응용 프로그램에서 이를 활용하고 있다. 컴퓨팅 환경을 통한 정보 이용 및 전달은 실생활에서 매우 편리한 서비스를 제공하고 있지만, 이를 악용하는 사례는 점차 증가하고 있으며 지능화되어가고 있다. 따라서, 컴퓨터에서 사용되는 정보들을 체계적으로 분석하고, 악의적인 정보의 유출을 효과적으로 차단하는 것은 안전한 컴퓨터 환경을 구성하기 위해서 반드시 해결해야 하는 과제이다.

소프트웨어가 가지는 취약성을 분석 및 보완하고 악의적인 정보 유출을 탐지하고 예방할 수 있는 연구의 필요성이 제기되고 있다. 소프트웨어 취약점을 분석하는 방법은 분석할 때, 프로그램의 실행 여부에 따라 프로그램을 실행하지 않고 코드를 분석하는 정적 분석[1]과 프로그램을 실행하면서 실행 중의 상태를 분석하는 동적 분석[2]으로 나눌 수 있다. 정적 분석은 분석 과정에서 소프트웨어를 실행하지 않기 때문에 코드의 분석을 통해서 전체 프로그램 영역을 분석에 반영할 수 있다.

반면 동적 분석 방법은 프로그램의 실행을 모니터링할 수 있기 때문에 정확한 실행 분석이 가능하지만 프로그램의 실행 영역에 대한 분석 결과만 반영된다는 단점이 있다. 본 논문에서는 소프트웨어의 정적 분석을 통해서 소프트웨어에서 사용하는 정보들의 신뢰도를 확인하고 실행 중에 나타날 수 있는 정보의 흐름을 추적하고, 중요한 보안 정보가 외부에 노출될 수 있는 취약점을 탐지하는 방법을 제안한다. 따라서, 소프트웨어에 유입되는 의심스런 정보가 보안 정보에 접근하는 것을 탐지하고 이를 사전에 예방함으로서 소프트웨어 보안을 향상시킬 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 소프트웨어의 보안을 강화하는 기존의 연구 사례를 소개한다. 3장에서는 정적 데이터 접근 분석 방법을 제안하고, 4장에서는 3장에서 제안한 분석 방법을 구현하고 검증한 결과를 보여준다. 5장에서 결론을 맺는다.

• First Author: Hyun-il Lim, Corresponding Author: Hyun-il Lim
*Hyun-il Lim(hilim@kyungnam.ac.kr), Dept. of Computer Engineering, Kyungnam University
• Received: 2015. 05. 22, Revised: 2015. 07. 20, Accepted: 2015. 08. 04.
• This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(NRF-2010-0024658).

II. Related Work

최근의 컴퓨터 환경에서 보안 분야[3]의 다양한 연구가 활발하게 진행되고 있으며, 악성코드 탐지[4]와 같은 응용 기술을 통해서 시스템 보안을 향상시키기 위한 연구가 진행되고 있다. 소프트웨어에서 사용되는 정보의 흐름을 분석하고 소프트웨어 안전성을 강화하기 위해 Newsome 등[5] 은 동적 오염 분석을 통해 오염된 정보의 동적 분석 방법을 제안하였다. 이 방법은 실행 시간에 프로그램의 동작과 정보의 흐름을 추적하면서 악의적인 입력(malicious input)을 통해 생길 수 있는 비정상적인 제어 흐름의 변경과 같은 보안 문제점을 찾아낼 수 있다. 분석 과정에서 소프트웨어의 실행을 통해서 동작을 분석하기 때문에 소스 코드와는 별개로 소프트웨어의 취약점을 통해 나타날 수 있는 외부의 침입을 감지하고 방어하는 분석 도구로 사용될 수 있다.

Cheng[6], Clause[7], Bai[8] 등에서 동적 오염 분석을 응용한 분석 결과를 제시하고 있으며, 악성 코드 탐지 등의 응용 사례에 대한 연구를 수행하였다. Cheng 등[6]은 데이터의 흐름을 추적하는 보안 시스템을 개발하였다. 이 방법에서는 직접 메모리 매핑(direct memory mapping)을 이용해서 실행 시간 오버헤드를 줄이고 성능을 향상시킬 수 있었으며, 버퍼 오버플로나 포맷 문자열 공격(format string attack)과 같은 여러 형태의 네트워크를 통한 공격을 감지할 수 있다. Clause 등[7]은 동적 오염 분석 방법을 이용한 범용 분석 프레임워크를 개발하였다. 이 분석 방법에서는 일반적으로 자료 흐름을 통해 전파되는 오염 정보 뿐만 아니라 제어 흐름을 통해 나타나는 오염 정보의 전파도 추적할 수 있다. Bai 등[8]은 네트워크 접속시에 나타날 수 있는 데이터 보안 문제를 검출할 수 있는 동적 분석 방법을 기술하였다. 이 방법은 x86 가상 머신인 QEMU 가상 머신에서 나타나는 데이터의 흐름을 모니터링 하면서 나타나는 프로그램의 제어 흐름을 바꿀 수 있는 프로그램의 취약점을 검출할 수 있다. Yin 등[9]은 동적 오염 분석을 이용해서 악성 코드를 탐지하기 위한 응용 기술을 제안하였다. 의심스러운 오염 데이터의 흐름을 추적하고 이를 바탕으로 테인트 그래프를 작성함으로써 악성코드를 탐지하는데 활용하고 있다. 기존의 분석들이 오염된 데이터가 전파되고 사용될 때 프로그램의 취약점을 공격할 수 있는지를 예측하지만, 테인트 그래프를 통해서 데이터 흐름을 표현하고, 사용자의 민감한 정보를 악용하려는 행동을 분석함으로써 악성코드를 찾아내는 데 이용하였다.

Table 1.은 관련 연구들을 요약하고 응용 사례들을 비교해서 보여준다. 본 논문에서는 정적 데이터 접근 분석을 통해서 사용자의 정보를 보호하는 알고리즘을 제시한다.

Table 1. Summary of related work.

Authors	Approach	Applications
Newsome et al [5]	Tracing flows of malicious data through dynamic taint analysis.	Software vulnerability analysis
Cheng et al [6]	Developing a security system for tracing data flows.	Network attack analysis
Clause et al [7]	Developing a generic analysis framework through dynamic taint analysis.	Taint analysis of data flows and control flows
Bai et al [8]	Developing a dynamic analysis for data security problems in network connections.	Detecting data security in network environments
Yin et al [9]	Detecting malware through dynamic taint analysis.	Malware detection

III. Static Data Access Analysis

1. The Definition of Base Language

정적 데이터 접근 분석을 위한 기반 언어는 절차적 프로그램에서 필수적인 명령어 구문을 포함하는 명령형 언어(imperative language)인 프로그래밍 언어 While[10,11]에 기반을 두고 구성하였다. 프로그래밍 언어 While은 프로그램의 실행에 필수적인 구문인 대입문(assignment), 순차구문(sequence), 조건문(conditional), 반복문(while)으로 구성된다. 본 논문에서는 While에 포함된 구문을 모두 포함하고 있으며, 분석 방법을 바이너리 프로그램에 확장할 수 있도록 메모리를 직접 다루는 접근 명령어인 load()와 store() 명령어를 지원하도록 확장하였다.

```
program ::= program begin  statement end
statement ::= skip |
              { statement } |
              statement; statement |
              identifier = exp |
              if exp then statement else statement |
              while exp do statement |
              identifier = load (exp) |
              store (exp, exp)
exp ::= identifier |
        true | false |
        constant |
        addr     |
        exp ⊗ exp |                  // binary operation
        ⊖ exp     |                  // unary operation
        input (data, src)
```

Fig. 1. The syntax of base language.

Fig. 1에서 본 논문에서 사용하는 기반 언어의 구문(syntax)을 보여주고 있다. 전체 프로그램의 구조는 명령어 구문(statement)으로 정의된다. 명령어 구문은 프로그램을 구성하

는 명령어들로 구성되며, 명령어의 시퀀스, 대입문, if 조건문, while 반복문 등 을 포함하고 있다. 또한, 메모리 접근 명령을 지원하기 위해서 메모리 로드(load) 및 스토어(store) 명령어를 포함한다. load 와 store 명령어는 메모리에서 데이터를 직접 읽어 오거나 쓰는 명령어이다.

표현식(exp)은 프로그램 명령어에 사용되는 수식을 나타내는데, 변수 이름을 표현하는 identifier, true 또는 false 값을 가지는 불린 값, 상수, 메모리 주소(addr), 이항 연산 및 단항 연산 등을 포함한다. 또한, 프로그램 외부에서 새로운 데이터를 입력 받을 수 있는 명령어 input()을 포함하고 있다. 이 명령어는 프로그램의 수행 중에 사용자나 네트워크 등의 외부로부터 정보를 입력 받는 연산을 수행한다.

2. The Design of Static Data Access Analysis

2.1 분석 환경 구성

정적 데이터 접근 분석을 수행하기 위해서 프로그램의 구문을 분석하는 과정을 추적하고 관리하는 분석 환경 정보의 구성이 필요하다.

Table 2. Analysis environment for static data access analysis.

Id_Env	It manages the analyzed values and properties of identifiers used in a program.
Mem_Env	It manages the stored values and properties of memory locations during program execution.
Property	It represents property of data according to the security levels. (Safe, Curious, Secure, Unknown, and Warning)

Table 3. Data property information for static data access analysis.

Property	Meaning
Safe	Safe data that can be trusted
Curious	Curious data originated from untrustworthy source
Secure	Secure data that should be kept securely
Unknown	Unknown data that is potentially curious
Warning	Warning for possibility of secure data exposure

Table 2에서 프로그램의 분석 상태를 표현하는 환경 정보들을 보여주고 있다. Id_Env는 프로그램에서 사용된 변수들의 정보를 분석한 결과를 저장하고 관리하는 역할을 한다. 각 변수에 대해서 결과값과 정보의 속성 정보를 유지하고 관리한다. 환경 변수 Mem_Env는 프로그램에서 사용되는 메모리의 상태와 변화 정보를 관리한다. 메모리 주소에 대해 저장된 값과 이 값의 속성 정보를 함께 관리한다. 위의 두 가지 정보는 분석 과정에서 나타나는 변수 및 메모리에 저장된 정보의 값과 데이터 속성 정보를 추적할 수 있는 환경을 제공한다. Property는 프로그램에서 사용되는 데이터의 속성 분석 정보 결과를 나타낸다. 이 값은 분석된 데이터의 신뢰도 속성에 따라서 Table 3과 같이 다섯 가지 값으로 분류된다.

Safe 데이터는 신뢰할 수 있는 안전한 값을 나타내고 Curious 데이터는 내용의 안전성을 신뢰할 수 없는 위험한 정보를 포함하는 데이터를 나타낸다. Secure 데이터는 외부로 유출되면 악용될 수 있는 보안 데이터를 나타낸다. Unknown 데이터는 정적 데이터 접근 분석을 통한 데이터 속성이 불분명한 경우를 나타내며 잠재적으로 의심(Curious) 데이터로 분류된다. Warning 데이터는 의심 데이터가 보안 데이터에 접근함으로써 정보 유출 및 악용될 수 있는 경우에 나타나며, 소프트웨어의 취약성을 나타내는 위험 상황을 표현한다.

2.2 표현식의 분석

프로그램에서 데이터의 이동과 접근 정보를 추적하고 분석하는 데이터 접근 분석은 프로그램을 구성하는 명령어에 대해서 명령어가 실행되면서 상태가 어떻게 변화되는지를 분석하고, 그 변화로부터 전파되는 의심 데이터의 전파 및 보안 데이터의 상태가 결정되어야 한다.

Fig. 2에서는 기반 언어에서 기술된 표현식(expression)에 대한 분석 알고리즘을 보여주고 있다. 표현식의 분석 함수 eval()은 분석하는 수식 exp와 분석에 필요한 환경 정보 Id_Env와 Mem_Env를 매개변수로 받아서 수식의 계산 결과값과 함께 수식의 데이터 속성 정보를 분석 결과로 반환한다.

```
eval(exp, Id_Env, Mem_Env) {
  switch (exp) {
    case identifier : return Id_Env(identifier);
    case true : return (true, Safe);
    case false : return (false, Safe);
    case constant : return (constant, Safe);
    case input(data, src) :
        if src is from safe source
            then return (data, Safe);
        else if src is from secure source
            then return (data, Secure);
        else if src is from curious source
            then return (data, Curious);
        else return (Unknown, Unknown);
    case addr :
        return(Mem_Env(addr));
    case ⊖ exp1 :
        (val, property) =
            eval(exp1, Id_Env, Mem_Env);
        return (⊖ val, property);
    case exp1 ⊗ exp2 :
        (val1, property1) =
            eval(exp1, Id_Env, Mem_Env);
        (val2, property2) =
            eval(exp2, Id_Env, Mem_Env);
        val = val1 ⊗ val2;
        prop = combine(property1, property2);
        return (val, prop);
  }
}
```

Fig. 2. Analysis algorithm of expressions.

수식이 변수(identifier) 인 경우 환경 정보 Id_Env의 분석

정보를 반환한다. 수식이 불린 값이거나 상수인 경우 프로그램 내에서 선언된 값이므로 데이터 속성값은 안전한 Safe로 분석한다. 데이터 입력 연산 input() 인 경우, 다양한 소스로 부터의 입력 데이터를 받아들인다. 이 때 입력 데이터의 소스의 신뢰도 정도에 따라서 안전한 데이터 인 경우 Safe, 보안 데이터 인 경우 Secure, 의심 데이터 인 경우 Curious 속성을 가진다. 또한 입력값이 정적 분석으로부터 분석이 불가능한 데이터는 Unknown 속성으로 분석하고 잠재적인 의심 데이터로 관리한다. 따라서, input() 명령에서 안전하지 않은 소스로 부터 유입된 정보는 Curious 속성으로 분석하며, 프로그램에서 의심 정보의 흐름을 분석할 수 있다. 수식이 메모리 주소 addr 인 경우 환경 정보 Mem_Env로부터 분석 결과를 반환한다.

수식이 단항 연산자 ⊖를 포함하는 단항식인 경우, 내부 수식 exp1의 데이터 속성을 물려 받는다. 그리고, 수식이 이항 연산자 ⊗를 포함하는 수식인 경우, 내부 수식 exp1과 exp2의 분석 결과를 결합(combine) 한 결과를 데이터 속성으로 가진다. 여기서 데이터의 결합은 의심 데이터가 보안 데이터에 접근하고 이를 유출하는 등 악의적인 연산에 사용될 수 있기 때문에 주의 깊게 분석할 필요가 있다.

Table 4. Data combine rule for property p1 and p2.

p1 \ p2	Safe	Secure	Curious	Unknown	Warning
Safe	Safe	Secure	Curious	Unknown	Warning
Secure	Secure	Secure	Warning	Warning	Warning
Curious	Curious	Warning	Curious	Curious	Warning
Unknown	Unknown	Warning	Curious	Unknown	Warning
Warning	Warning	Warning	Warning	Warning	Warning

Table 4는 두 데이터 속성 p1과 p2에 대해서 연산을 통해 함께 사용되는 경우 분석 결과를 나타내는 결합 규칙(combine rule)를 보여준다. 이 결합 규칙은 소프트웨어의 안전성을 검증하기 위해서 의심 데이터가 전달되는 것을 분석하고 전파되는 과정을 나타내고 있으며, 의심 데이터가 보안 데이터에 접근하는 경우에 Warning의 데이터 속성으로 분석함으로써 보안 데이터의 비정상적인 사용을 분석할 수 있다. 데이터 결합 규칙에서 Warning이 발생하는 경우는 Secure 정보가 Curious 정보 또는 Unknown 정보에 의해서 접근될 때 발생한다. 예를 들어, 프로그램에서 의심 데이터로 분석된 이메일 주소를 포함하는 정보가 신용카드 번호와 같은 보안 데이터에 접근하는 경우, 신용카드 번호가 해당 메일 주소를 통해유출될 위험성이 있다. 따라서 의심 데이터의 접근을 Warning 데이터로 분석한다.

2.3 프로그램의 데이터 접근 분석

본 절에서는 기반 언어의 명령어에 대해서 데이터 접근 분석 과정을 제안한다. Fig. 3은 본 논문에서 제안하는 데이터 접근 분석 알고리즘을 보여주고 있다. analysis() 는 분석하는 명령어 stmt와 분석에 필요한 환경 정보 Id_Env, Mem_Env를 매개변수로 받고, 분석 결과를 환경 정보 Id_Env와 Mem_Env를 통해 반환한다.

```
analysis(stmt, Id_Env, Mem_Env) {
  switch (stmt) {
    case skip : return (Id_Env, Mem_Env);
    case {stmt} :
         return (analysis(stmt, Id_Env, Mem_Env));
    case (stmt1; stmt2) :
         (Id, Mem) = analysis(stmt1, Id_Env, Mem_Env);
         return(analysis(stmt2, Id, Mem));
    case var = exp :
         (val, p) = eval(exp, Id_Env, Mem_Env);
         Id_Env = Id_Env + (var : (val, p));
         return (Id_Env, Mem_Env);
    case if exp then stmt1 else stmt2 :
         (val, property) = eval(exp, Id_Env, Mem_Env);
         if val == true then
            return analysis(stmt1, Id_Env, Mem_Env);
         else if val == false then
            return analysis(stmt2, Id_Env, Mem_Env);
         else if val == Unknown then
            (Id1,Mem1)=analysis(stmt1,Id_Env,Mem_Env);
            (Id2,Mem2)=analysis(stmt2,Id_Env,Mem_Env);
            return Union of two environment infos;
    case while exp do stmt1:
         (val, property) = eval(exp, Id_Env, Mem_Env);
         if val == true then
            (Id, Mem) = analysis(stmt1, Id_Env,Mem_Env);
            return analysis(while exp do stmt1, Id, Mem);
         else if val == false then
            return ( Id_Env, Mem_Env) ;
         else if val == Unknown then
            (Id1,Mem1) = analysis(stmt1,Id_Env,Mem_Env);
            (Id2,Mem2) = analysis(stmt1, Id1, Mem1);
            return Union of the environment infos;
    case var = load(exp) :
         (addr, p1) = eval(exp, Id_Env, Mem_Env);
         (val, p2) = Mem_Env(addr);
         Id_Env = Id_Env + (var : (val, combine(p1, p2)));
         return (Id_Env, Mem_Env);
    case store(exp1, exp2) :
           (addr, p1) = eval(exp1, Id_Env, Mem_Env);
         (val, p2) = eval(exp2, Id_Env, Mem_Env);
         Mem_Env=Mem_Env+ addr:(val,combine(p1, p2)));
         return (Id_Env, Mem_Env);
  }
}
```

Fig. 3. Data access analysis algorithm for statements.

skip 명령어의 경우는 특별한 분석이 필요없이 현재의 분석 결과를 그대로 반환한다. 명령어의 시퀀스 인경우는 첫 번째 명령어에 대해서 분석한 결과를 받고 이 결과를 순차적으로 다음 연산에 반영하면서 분석을 수행한다. 대입문은 수식의 연산 결과를 변수에 대입하는 명령어이다. 따라서 eval() 을 이용해서 수식 exp를 분석한 결과를 변수 var에 대입하는 연산을 수행하게 되고, 분석 결과로 변수 var에 대한 결과값과 데이터 속성 정보는 Id_Env에 추가된다.

if 조건문인 경우 조건식이 참인 경우 stmt1의 분석 결과를 반환하고, 조건식이 거짓인 경우 stmt2의 분석 결과를 반환한다. 조건식의 값이 정적 분석을 통해서 결정되지 않는 Unknown 인 경우는 실제 실행 중에 참이 되는 경우와 거짓이 되는 경우를 모두 고려해야 안전한 결과를 얻을 수 있다. 따라서, stmt1과 stmt2를 분석한 결과를 모은 최종 결과를 반환한다. 이는 정적 분석에서 추론되지 않는 조건식에 대해서도 발생할 수 있는 가능한 실행 조건을 함께 반영함으로써 데이터 접

근 분석의 정확도를 높일 수 있다.

while 반복문인 경우 조건식이 참인 경우는 반복문을 계속 수행하기 때문에 반복문의 바디 stmt1을 분석한 결과 환경 정보를 이용하여 재귀 반복 분석한다. 조건식이 거짓인 경우는 반복문을 종료하고 현재의 분석 환경 정보를 그대로 반환하면 된다. 반면에 조건식의 값이 정적 분석을 통해서 분석되지 않는 경우는 반복문의 다양한 실행 경로를 모두 고려한 분석을 해야 한다. 반복문의 검증기준에 기초해서 분석을 시도하고, 반복문의 반복 회수가 0회, 1회, 1회 이상인 경우에 대해서 분석한 결과를 모아서 최종 결과를 반환함으로서 가능한 다양한 실행 경우에 대한 분석 정보를 반영한다.

load 명령문인 경우 수식 exp를 분석한 결과로부터 메모리의 주소를 계산하고, 환경 정보 Mem_Env로부터 해당 주소에 저장된 정보를 읽어온다. 메모리의 결과값은 대입문의 분석 결과를 반영하기 위해서 환경 정보 Id_Env에 반영한다. 데이터 속성값은 메모리의 주소에 대한 속성과 메모리에 저장된 값의 속성을 결합한 결과값을 속성값으로 반환한다. store 명령문인 경우 메모리에 값을 저장하기 위해 수식 exp1 으로부터 주소값을 분석하고, 수식 exp2 로부터 저장값을 분석한다. 메모리에 스토어되는 분석 결과를 반영하도록 환경 정보 Mem_Env에 저장되는 정보를 수정한 후 환경 정보들을 반환한다.

본 논문에서 제안한 데이터 접근 분석은 프로그램의 명령문으로부터 여러 데이터가 사용되고 사용되는 값이 전파되는 과정을 분석한다. 각 데이터는 정보의 신뢰도 정도에 따라서 데이터 속성값을 분석하고, 분석된 결과로부터 의심 데이터의 전파 및 보안 데이터가 노출될 수 있는 위험한 상황을 감시하고 소프트웨어의 실행 안전성을 향상 시킬 수 있다.

IV. Implementation and Evaluation

본 장에서는 본 논문에서 제안한 정적 데이터 접근 분석의 구현 및 실험 결과를 보여준다. 분석기는 기반 언어에 적용할 수 있으며, 테스트 프로그램을 통해 분석 결과를 보여준다.

본 논문에서 제안한 정적 데이터 접근 분석 알고리즘의 개발 환경은 Microsoft Windows 7 운영체제에서 함수형 프로그래밍 언어 Haskell[13]을 이용해서 구현하였다. 분석기는 기반 언어로 작성된 프로그램으로부터 프로그램의 데이터들의 신뢰도에 따라 속성값을 분석하고 의심 데이터의 전파 및 보안 데이터의 안전성을 분석하고 결과를 출력한다.

Fig. 4는 기반 언어를 이용해서 구현된 분석기에 대한 입력 테스트 프로그램을 보여준다. 이 프로그램은 기반 언어에서 포함하고 있는 skip, 대입문, input() 명령문, if 조건문, while 반복문, 메모리 load/store 등 기반 언어의 모든 명령어와 수식을 포함하고 있으며, 모든 분석 규칙이 포함될 수 있도록 구성되었다. 본 프로그램은 보안 데이터와 의심 데이터인 이메일 주소를 입력으로 받은 후, 데이터의 흐름을분석하며, 반복문은 0에서 100까지의 합을 반복적으로 계산하는 프로그램으로 구성하였다.

```
    program begin
 1:   a = 100;
 2:   s = input( SecureNum,
            Secure number from user input);
 3:   x = input( "a@b.mail",
                  Curious e-mail address);
 4:   addr = a;
 5:   store( addr, x);
 6:   if ( s >= 0 ) then s = s ⊗ x;
 7:                 else store (a, s);
 8:   b = load(a);
 9:   b = b ⊗ x;
10:   skip;
11:   t = input( "SafeData", safe value for init.);
12:   i = 0;
13:   j = -10;
14:   k = input("-1", value from unknown source);
15:   while (i <= 100) do
16:       j = j + i;
17:       i = i + 1;
18:   store( i, j);
19:   i = t ⊗ k;
    end
```

Fig. 4. A test program

Table 5는 테스트 프로그램에 대한 정적 데이터 접근 분석 과정을 보여주고 있다. 프로그램의 각 명령어에 대해서 분석 결과가 수행됨에 따라서 환경 정보 Id_Env와 Mem_Env에 나타나는 분석 정보를 단계적으로 보여준다.

Table 5. The analysis procedure for the test program.

	Id_Env	Mem_Env
1	a (100, Safe)	
2	s (SecureNum, Secure)	
3	x ("a@b.mail", Curious)	
4	addr (100, Safe)	
5		100 ("a@b.mail", Curious)
6	s (SecureNum ⊗ "a@b.mail", **Warning**)	
7		100 (SecureNum ⊗ "a@b.mail", **Warning**)
8	b (SecureNum ⊗ "a@b.mail", Warning)	
9	b (SecureNum ⊗ "a@b.mail" ⊗ "a@b.mail", **Warning**)	
10		
11	t ("SafeData", Safe)	
12	i (0, Safe)	
13	j (-10, Safe)	
14	k ("-1", Unknown)	
15		
16	j (5040, Safe)	
17	i (101, Safe)	
18		101 (5040, Safe)
19	i ("SafeData" ⊗ "-1", Unknown)	

라인 1, 2, 3에서는 속성이 다른 세 개의 데이터가 설정된다. 변수 s는 사용자로부터 보안 데이터(비밀번호, 주민번호 등)을 입력으로 받고 속성은 Secure 값으로 나타나며, 변수 x는 외부로부터 유입되는 의심스런 이메일 주소를 가진다. 라인 4에서

는 상수 a가 addr 변수에 대입되고, 라인 5에서는 메모리의 addr 번지에 변수 x의 데이터가 저장된다. 따라서 메모리 환경 정보 Mem_Env에는 100번지에 Curious 속성을 가진 메일 주소가 저장된다.

라인 6에서는 if 조건문의 조건식에 따라서 실행 구문이 달라진다. 조건식은 사용자로부터 입력 받은 보안 데이터 s의 값에 따라서 결과가 달라지지만 정적 분석에서는 사용자의 입력에 대한 참 거짓을 판단할 수 없으므로 조건식의 결과는 Unknown이 된다. 따라서 실행 가능성이 있는 두 개의 if 구문의 바디가 한 번씩 분석된다. 라인 6에서는 보안 데이터 s와 의심 데이터 x가 함께 사용되고 있기 때문에 의심 데이터가 보안 데이터에 접근하고 있으며, 보안 데이터의 유출 가능성을 나타내는 Warning 속성을 가진다. 라인 7에서는 보안 정보 s의 값이 메모리 100번지에 저장되는데 이 때 메모리에 저장된 의심 데이터에 접근하게 되면서 Warning 이 나타난다.

라인 8은 메모리 100번지에서 값을 읽어서 변수 b에 대입하고 라인 9에서는 변수 b에 저장된 값이 의심 변수 x에 접근하면서 Warning 이 발생한다. 라인 10, 11에서 skip 명령어는 실행하지 않으며, 다음 input()명령어는 안전한 값 “SafeData”를 초기값으로 입력받아서 변수 t에 대입한다. 라인 12~18은 while 반복문을 검증하기 위해서 변수 i와 j를 각각 0과 -10으로 초기화하고 0에서 100까지 i의 합을 구한 후 i번지에 결과를 저장(store)한다. 따라서 메모리 101번지에는 5040이란 값이 저장되고 속성은 Safe로 분석된다. 라인 19에서는 Safe 변수 k에 14라인에서 미지의 소스로부터 유입된 Unknown 변수 t가 접근하고 있으며 결과가 변수 i에 대입된다. 따라서 변수 i의 속성값은 Unknown이 된다.

본 분석 과정에서 의심 데이터인 메일 주소는 총 3번 보안 데이터에 접근하고 있으며, Warning 속성이 나타남을 알 수 있다. 이는 주민 번호 같은 보안 데이터가 검증되지 않은 데이터인 이메일 주소에 의해 유출되는 등의 위험성을 예측할 수 있다.

Table 6. The analysis results of the test program.

Mem. loc.	Value	Property
100	SecureNum ⊗ “a@b.mail”	**Warning**
101	5040	Safe
Identifier	**Value**	**Property**
a	100	Safe
s	SecureNum ⊗ “a@b.mail”	**Warning**
x	“a@b.mail”	Curious
addr	100	Safe
b	SecureNum ⊗ “a@b.mail” ⊗ “a@b.mail”	**Warning**
t	“SafeData”	Safe
i	“SafeData” ⊗ “-1”	Unknown
j	5040	Safe
k	“-1”	Curious

Table 6은 데이터 접근 분석을 통한 테스트 프로그램의 분석 결과를 보여주고 있다. 분석 결과에서 프로그램의 메모리 주소와 변수에 대해서 분석을 통해 나타나는 결과값과 신뢰도 속성을 보여주고 있다. 결과로부터 메모리 100번지와 변수 s와 b가 Warning 속성을 가지는 것을 확인할 수 있다.

Fig. 5는 구현한 분석기로 테스트 프로그램을 분석한 결과를 보여준다. 분석 결과에서 프로그램의 수행 중에 나타날 수 있는 보안 데이터의 유출 가능성을 감지하고 3개의 경고 메시지를 보여주고 있다. 각 경고 메시지는 프로그램의 보안 데이터인 SecureNum 입력 정보가 의심 데이터인 메일 주소 “a@b.mail” 주소와 함께 사용될 때 보안 정보의 유출 가능성을 감지하고 경고 메시지를 보인다는 것을 알 수 있다. 따라서 분석 결과는 앞서 보여준 단계별 분석 과정에서 나타난 결과와 일치하고 있음을 확인할 수 있다. 그리고, 분석이 끝난 후 Mem_Env와 Id_Env 환경 정보의 분석 결과를 함께 출력하고 있으며, 이는 Table 5에서 단계별 분석 과정의 최종 분석 결과와 일치함을 확인할 수 있다.

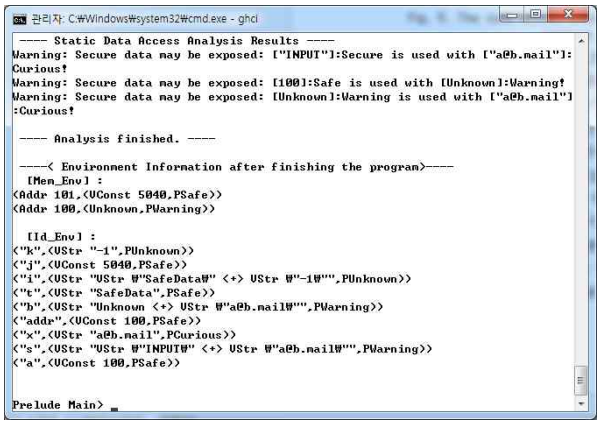

Fig. 5. The data access analysis execution result for the test program.

분석 결과로부터 프로그램의 데이터 속성에 따라 의심 데이터의 전파를 효과적으로 분석할 수 있으며, 보안 데이터에 접근 가능성을 감지하고 이를 경고함으로써 보안 데이터 유출 위험성이 있는 소프트웨어의 취약점을 분석할 수 있다.

V. Conclusion

소프트웨어의 광범위한 활용과 더불어 소프트웨어 취약점에서 나타나는 보안 문제는 점점 더 중요한 문제가 되고 있다. 정적 데이터 접근 분석은 소프트웨어에 사용되는 보안 정보와 의심 정보의 흐름을 분석하고 의심 정보가 보안 정보에 접근하는 과정을 탐지한다. 이는 보안 정보의 유출로 인해 나타날 수 있는 다양한 피해 사례를 사전에 예방할 수 있는 시스템을 구축하는 데 중요한 역할을 할 수 있을 것이다.

본 논문에서는 정적 데이터 접근 분석의 알고리즘을 설계하고 기반 언어에 대한 분석기를 구현하였다. 구현한 결과는 테스트 프로그램을 통해서 결과를 확인하였고, 의심 데이터의 흐름

과 보안 데이터에 접근 과정을 정확하게 분석할 수 있었다. 본 논문의 분석 알고리즘은 정보의 흐름을 통한 소프트웨어의 취약점을 분석함으로써 시스템 및 소프트웨어의 보안을 강화하기 위한 환경을 제공할 수 있으며, 소프트웨어의 정적 데이터 분석을 위한 기반 기술로 활용될 수 있을 것이다.

REFERENCE

[1] Marco Pistoia, Satish Chandra, Stephen J. Fink, and Eran Yahav, "A survey of static analysis methods for identifying security vulnerabilities in software systems," IBM Systems Journal, Vol. 46, No.2, pp. 265-288, 2007.

[2] Manuel Egele, Theodoor Scholte, Engin Kirda, Christopher Kruegel, "A Survey on Automated Dynamic Malware Analysis Techniques and Tools," ACM Computing Surveys, Vol. 44, No.2, Feb. 2012.

[3] Jeong-Hoon Jeon, "A study on the classification systems of domestic security fields," Journal of the Korea Society of Computer and Information, Vol. 20, No. 3, pp. 81-88, March 2015.

[4] Eun-Gyoem Jang, Sang Jun Lee, Joong In Lee, "A Study on Similarity Comparison for File DNA-Based Metamorphic Malware Detection," Journal of the Korea Society of Computer and Information, Vol. 19, No. 1, pp. 85-94, Jan. 2014.

[5] James Newsome and Dawn Song, "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software," In Proceedings of the Network and Distributed System Security Symposium, 2005.

[6] Winnie Cheng, Qin Zhao, Bei Yu, and Scott Hiroshige, "TaintTrace: Efficient Flow Tracing with Dynamic Binary Rewriting," In Proceedings of 11th IEEE Symposium on Computers and Communications, pp. 749-754, June 2006.

[7] James Clause, Wanchun Li, and Ro Orso, "Dytan: A Generic Dynamic Taint Analysis Framework," In Proceedings of the International Symposium on Software Testing and Analysis, pp. 196-206, 2007.

[8] Zhiwen Bai, Liming Wang, Jinglin Chen, Lin Xu, Jian Liu, and Xiyang Liu, "DTAD: A Dynamic Taint Analysis Detector for Information Security," In Proceedings of The Ninth International Conference on Web-Age Information Management, pp. 591-597, July 2008.

[9] Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda, "Panorama: capturing system-wide information flow for malware detection and analysis," In Proceedings of the 14th ACM conference on Computer and communications security, pp. 116-127, 2007.

[10] Matthew Hennessy, "The WHILE programming language," *https://www.cs.tcd.ie/Matthew.Hennessy/splexternal2015/notes/WhileSlides2to1.pdf*, 2015.

[11] Kenneth Slonneger and Barry L. Kurtz, "*Formal Syntax and Semantics of Programming Languages*," Addison Wesley, 1995.

[12] Sanjiva Prasad and S. Arun-Kumar, "An Introduction to Operational Semantics," In the Compiler Design Handbook: Optimizations and Machine Code Generation, pp. 841-890, CRC Press, 2002.

[13] Programming Language Haskell, *http://www.haskell.org/*

Authors

Hyun-il Lim received his B.S., M.S., and Ph.D. degrees in computer science from KAIST, Korea, in 1995, 1997, 2009, respectively.

Dr. Lim joined the faculty of the Department of Computer Engineering at Kyungnam University, Changwon, Korea, in 2010. He is currently an assistant professor in the Department of Computer Engineering, Kyungnam University. He is interested in software security, software protection, watermarking, and program analysis.