http://dx.doi.org/10.14400/JDC.2016.14.6.253

SAT를 이용한 정보흐름의 안전성 분석

김제민*, 고훈준**
인하대학교 컴퓨터공학과*, 경인여자대학교 영상방송과**

Security Analysis of Information Flow using SAT

Je-Min Kim*, Hoon-Joon Kouh**
Dept. of Computer and Information Engineering, Inha University*
Dept. of Video Broadcasting, Kyung-In Women's University**

요 약 PC와 모바일 기기에 있는 다양한 프로그램을 이용하여 인터넷을 이용하는 사람들이 늘어날수록 프로그램에서 개인정보 등이 유출될 가능성은 매우 높아지고 있다. 따라서 인터넷을 사용하는 프로그램에서 정보흐름의 안전성 분석을 한 후에 개인정보의 유출이 없는 안전한 프로그램을 사용해야 한다. 정보흐름의 안전성 분석은 프로그램 내에서 정보의 흐름이 안전한지 분석하는 방법으로 정보흐름이 안전하면 개인정보 유출이 없고 안전하지 않으면 개인정보 유출이 발생할 수 있다. 본 논문에서는 SAT 해결기를 활용하여 정보흐름 분석을 수행하는 방법을 제시한다. 이 방법은 보안 수준이 설정된 변수를 포함하는 프로그램을 제어와 정보흐름을 나타내는 명제 논리식으로 변환하고, SAT 해결기를 이용해 명제 논리식으로부터 만족가능성 여부를 판단한다. 판단된 결과를 통해 프로그램에서 정보흐름이 안전한지 알 수 있으며, 안전하지 않은 경우 반례를 생성하여 어느 부분에서 안전하지 않은 지 알 수 있다.

주제어 : 정보흐름분석, SAT 해결기, 반례, 명제 논리식, SSA 형태

Abstract As many people use internet through the various programs of PC and mobile devices, the possibility of private data leak is increasing. A program should be used after checking security of information flow. Security analysis of information flow is a method that analyzes security of information flow in program. If the information flow is secure, there is no leakage of personal information. If the information flow not secure, there may be a leakage of personal information. This paper proposes a method of analyzing information flow that facilitates SAT solver. The method translates a program that includes variables where security level is set into propositional formula representing control and information flow. The satisfiability of the formula translated is determined by using SAT solver. The security of program is represented through the result. Counter-example is generated if the program is not secure.

Key Words : Information Flow Analysis, Boolean Satisfiability Solver, Counter-Examples, Propositional Logic Formula, Static Single Assignment Form

Received 25 April 2016, Revised 24 May 2016
Accepted 20 June 2016, Published 28 June 2016
Corresponding Author: Hoon-Joon Kouh
(Kyung-In Women's University)
Email: hjkouh@kiwu.ac.kr

ISSN: 1738-1916

1. 서론

스마트폰의 열풍으로 PC 환경뿐만 아니라 모바일 환경에서 인터넷을 사용하는 사람들이 매우 많아졌으며 다양한 앱들을 통해 개인정보를 사용하고 있다. 대다수의 사람들은 앱들이 정보 누출에 안전한지 알지 못하며 불안감을 가지고 있다[3,18,19].

앱에서 개인정보가 누출하는지에 대해서는 정보흐름 분석으로 알 수 있다. 정보흐름 분석은 보안 수준을 여러 단계로 정의하고, 프로그램의 각 변수에 대해 적합한 보안 수준을 배정한 후 높은 보안 수준을 갖는 변수로 부터 낮은 보안 수준을 갖는 변수로 정보가 흐르는지 분석하는 것이다. 만약 보안 수준이 높은 변수에서 보안 수준이 낮은 변수로 정보의 흐름이 발생한다면 위험한 정보흐름이 존재하며 정보의 누출이 발생할 수 있다.

정보흐름의 안전성 분석 방법은 70년대 Denning이 프로그래밍 언어의 구문 수준에서 검사하는 방법[5]과 Lattice 모델을 이용해 여러 보안 수준을 나타내는 방법[4]을 시작으로 많은 연구가 이루어져 왔다[1]. 프로그램 언어 수준에서 정보흐름의 안전성 연구는 데이터 흐름분석(data flow analysis)[6], 타입 시스템(type system)[7,8] 기반의 분석, 요약 해석(abstract interpretation)[17] 등으로 다양하게 연구되어 왔다. 데이터 흐름분석을 이용하는 방법은 최적화와 정적 분석에 사용되는 분석 방법을 정보흐름 분석에 이용한다[6]. 타입 시스템을 이용하는 방법은 보안 수준을 타입으로 간주하고 안전한 정보흐름이 발생할 수 있도록 타입 시스템을 구성한 후 타입 검사를 하여 안전한 정보흐름이 발생하는지 검사한다. 의미론에 기반을 둔 방법은 타입을 기반으로 한 분석의 정확성을 높이기 위해 좀 더 세밀한 분석을 의미론에 기반하여 수행한다[9, 10]. 정적 분석이나 타입 시스템을 이용하는 방법 뿐 만 아니라 최근에는 논리에 기반을 둔 방법들이 제안되었다. Hoare-like 논리를 이용하는 방법으로서 간단한 명령형 언어에서 논리적으로 정보흐름을 표현하는 방법이 제안되었다[11]. 검증조건을 생성하는 방법[12]은 프로그램에 대한 검증 조건을 생성하여 정리 증명기를 활용할 수 있게 한다.

이러한 방법들은 안전한 프로그램을 안전하지 않다고 분석하는 경우가 너무 많아 정밀성이 떨어지며 다양한 프로그램 특성에 따라 설계가 복잡하고 어렵다.

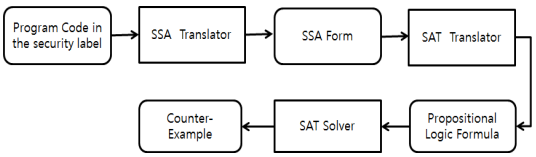
본 논문은 프로그램의 오류를 찾아내는데 사용되고 있는 SAT 해결기(boolean satisfiablity solver)[13,16]를 정보흐름 분석에 적용하여 안전성을 검사하는 방법을 제안한다. 이 방법은 논리식의 만족가능(satisfiable) 여부를 판별함으로써 프로그램이 안전한 정보흐름을 가지는지, 아니면 위험한 정보흐름을 가지는지를 분석한다. 만약, 프로그램이 안전하지 않은 정보흐름을 가지는 경우에는 반례(counter-examples)를 생성하여 안전하지 않음을 알려주어 분석의 정밀성을 높일 수 있다.

논문의 구성은 다음과 같다. 2장에서는 본 논문에서 제안하는 SAT 해결기를 이용하여 정보흐름의 안정성을 분석하는 방법에 대해 설명한다. 3장에서는 분석에 필요한 명제논리식을 구문 구조를 제시한다. 4장에서는 명제논리식을 생성하는 방법을 설명한다. 5장에서 예제를 통해 본 논문에서 제시한 방법을 확인한다. 마지막으로 6장에서 결론을 맺는다.

2. 정보흐름의 안전성 분석

보안성이 높은 정보를 누출하는 프로그램을 정보흐름의 안전성이 낮다고 말한다. 정보의 누출은 명시적 누출과 묵시적 누출로 나눌 수 있다[14]. 변수 l이 보안수준 low, 변수 h가 보안수준 high일 때 [l:=h]와 같은 배정문은 높은 보안 수준을 갖는 변수인 h로 부터 낮은 보안 수준을 갖는 변수 l로 값이 배정되므로 정보 누출이 발생하고 이러한 경우를 명시적 정보 누출이라고 한다. 묵시적 정보 누출은 if문이나 while 문에서 발생할 수 있는데, 예를 들어 변수 p가 높은 보안 수준을 가질 때 if [p=1] then [z:=1] else [z:=0]과 같은 if문의 경우 z의 값에 따라 높은 보안 수준을 갖는 변수 p의 값이 1인지 아닌지 알 수 있으므로 이러한 경우를 묵시적 정보 누출이라고 한다.

본 논문에서는 정보 누출의 여부를 찾기 위해서 정보흐름 분석으로 프로그램의 안전성을 확인하는 방법으로 SAT 해결기를 이용하는 방법을 제안한다. SAT 해결기를 이용하여 정보흐름의 안전성을 분석하기 위해서는 [Fig. 1]과 같이 프로그램을 SSA 형태(static single assignment form)로 변환하고, 다시 명제 논리식(propositional logic formula)[15]으로 변환한다.



[Fig. 1] Process of Secure Information Flow Analysis using SAT

[Fig. 1]에서 SSA Translator는 R. Cytron[2]이 제안한 방법으로 프로그램 코드를 SSA 형태의 코드로 변환한다. SAT Translator는 SSA 형태의 코드를 명제 논리식으로 변환하고, SAT 해결기를 이용하여 명제 논리식의 만족가능(satisfiable) 여부를 판별하여 프로그램이 안전한 정보흐름을 가지는지, 아니면 위험한 정보흐름을 가지는지를 분석한다.

정보흐름 분석을 설명하기 위해 [Fig. 2]와 같이 정보 수준을 명시할 수 있는 선언문과 print문이 추가된 간단한 While 프로그래밍 언어를 사용한다.

Program	$prog \in Program$
	$prog ::= dec;s$
Variable	$dec \in Dec$
Declaration	$dec ::= high\, x := a; dec \mid low\, x := a; dec$
	$\mid unknown\, x := a; dec \mid \varepsilon$
Statement	$s \in Stmt$
	$s ::= x := a \mid skip \mid s_1; s_2 \mid print\, a$
	$\mid \text{if } c \text{ then } s_1 \text{ else } s_2 \mid \text{while } c \text{ do } s$
Variable	$x \in Var$
Arithmetic	$a \in AExp$
Expression	$a ::= n \mid x \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2$
Conditional	$c \in CExp$
Expression	$c ::= true \mid false \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg c \mid c_1 \wedge c_2$

[Fig. 2] Syntax of While Programming Language

구문 prog는 변수 선언 부분과 문장 부분으로 나누어진다. 선언 부분 dec는 문장에서 사용되는 변수를 선언하며, 이 때 변수의 보안 수준을 설정한다. 보안 수준은 여러 단계로 나눌 수 있지만 본 논문에서는 높은 수준의 경우 high, 낮은 수준의 경우 low, 보안수준을 알 수 없을 경우 unknown으로 설정한다. 예를 들어, high h:=10의 의미는 변수 h의 보안 수준은 high이고 초기의 배정된 값은 10이라는 것이다. 문장 s는 배정문, skip 문, 문장의 결합, if 문, while 문, 산술 표현식의 값을 출력하기 위한 문장인 print문 등으로 구성된다. 정보 누출은 주로 print 문에서 발생한다. print문에서 출력되는 산술 표현식의 보안 수준에 따라 정보 누출의 여부를 확인할 수 있다. 산술 표현식 a는 기본적인 정수형 값이나 변수, 또는 산술 표현식들 간의 연산을 나타낸다. 산술 표현식에는 +, -, *, / 등의 연산자가 포함될 수 있다. 조건 표현식 c는 if 문이나 while 문에서 조건 판단을 위해서 사용된다. 조건 표현식 c는 true, false 와 같은 진리값과 산술 표현식 간의 관계 연산이나 조건 표현식 간의 불리언 연산을 포함할 수 있다.

[Fig. 2]의 While 언어 구문으로 부터 간단한 프로그램을 작성하면 [Fig. 3]과 같다.

```
1  [high h:=10];
2  [low l:=0];
3  [unknown p:=10];
4  [low t:=1];
5  [unknown e:=0];
6  [low z:=0];}
7  [l:=h+t];
8  if [p=1]
9  then {[z:=100];[z:=t]}
10 else {[z:=e]};
11 [print z];
```

[Fig. 3] While Language Program Example

[Fig. 3]의 예제 프로그램을 SSA Translator를 사용하여 SSA 형태로 변환하면 [Fig. 4]와 같다.

```
1  [high h0:=10];
2  [low l0:=0];
3  [unknown p0:=10];
4  [low t0:=1];
5  [unknown e0:=0];
6  [low z0:=0];}
7  [l1:=h0+t0];
8  if [p0=1]
9  then {[z1:=100];
         [z2:=t0]}
10 else {[z1:=e0];
         [z2:=z1]};
11 [print z2]
```

[Fig. 4] SSA Form

3. 정보흐름을 분석을 위한 명제 논리식

SAT 해결기로 정보흐름 분석을 다루기 위해서는

[Fig. 5]의 명제 논리식 구문을 사용하여 SSA 형태의 프로그램을 명제 논리식으로 모델링해야 한다.

$$p \in \mathrm{Pr}oposition ::= x \mid T \mid F \mid x_c$$
$$f \in Formula ::= f_1 \leftrightarrow f_2 \mid f_1 \rightarrow f_2 \mid f_1 \wedge f_2 \mid f_1 \vee f_2 \mid \neg f \mid p$$

[Fig. 5] Syntax of Propositional Logic Formula

명제 논리식의 연산자의 우선순위는 ¬가 가장 높고 ∧, ∨, →, ↔ 순으로 높다. x는 논리식 변수로 프로그램 변수에 대응된다. x_c의 경우 조건 표현식을 논리식의 변수로 변환한 형태이다.

변수의 보안 수준을 다룰 때는 보안 수준이 높을 경우 진리값 중 참 값인 T로 모델링되고 보안 수준이 낮을 경우 거짓 값인 F로 모델링된다. 선언되는 변수 x의 보안수준에 따라 보안수준이 high일 경우 x↔T, low일 경우 x↔F와 같은 논리식으로 표현한다.

배정문에 대해서는 배정문의 := 기호를 기준으로 왼쪽을 lhs(left hand side)로 오른쪽을 rhs(right hand side)라고 하면 rhs가 높은 보안수준을 갖고 lhs가 낮은 보안수준을 가질 때만 정보흐름이 안전하지 않다. 그런데 rhs는 산술식으로서 보안 수준이 다양한 여러 개의 변수를 이루어질 수 있다. 그럴 경우 여러 개의 변수 중 보안 수준이 높은 변수가 rhs의 보안수준을 결정한다. 이를 논리식 함축(implication)으로 표현하면 <Table 1>와 같다. 예를 들어 x:=y+z와 같은 배정문에 대해서는 y+z의 보안수준은 논리식 y∨z로 나타낼 수 있고, 배정문은 논리식 y∨z→x로 나타낼 수 있다.

<Table 1> Modeling of information flow for assignment statement using implication

lhs	rhs	rhs→lhs	Security
T(High)	F(Low)	T(High)	O
F(Low)	T(High)	F(Low)	X
T(High)	T(High)	T(High)	O
F(Low)	F(Low)	T(High)	O

$s_1;s_2$와 같은 문장의 연속의 경우 s_1에서 생성된 논리식과 s_2에서 생성된 논리식이 동시에 만족해야 한다. 따라서 s_1으로부터 생성된 논리식을 f_1, s_2로부터 생성된 논리식을 f_2라 할 때, $s_1;s_2$를 위한 논리식은 $f_1 \wedge f_2$다. skip 문장의 경우 정보흐름이 발생하지 않으므로 논리식으로 변환이 필요 없다.

print a 문의 경우에는 a가 변수이면서 high를 가지면 안 되므로 a→F와 같은 논리식이 생성된다. a가 x와 y를 구성하는 문장의 경우 산술 표현식 a로부터 보안수준이 낮은 표준출력으로 정보흐름이 발생하는지 확인해야 한다. 배정문과 유사하게 산술식 a를 구성하는 변수 중 보안수준이 높은 변수가 산술식 a의 보안 수준을 결정한다. 따라서 print a의 경우 x∨y→F와 같은 논리식이 생성된다.

if문의 경우 묵시적 정보 누출과 명시적 누출의 경우를 구분하여 논리식으로 생성함으로써 정보흐름이 안전한지 검사해야 한다. if 문에서 묵시적 정보 누출은 조건문의 변수 중 하나라도 보안 수준이 high일 때 참, 거짓에 상관없이 발생하므로 then과 else절의 정보흐름을 표현해야한다. if 문의 조건절의 변수와 then절과 else절의 배정문 중 lhs로의 정보흐름을 검사해야 한다. [Fig. 4]의 경우, if 문의 조건절의 변수중 하나라도 높은 보안수준을 가지는 논리식은 ¬(p_0↔F)로 표현된다. then절을 위한 논리식은 F→z_1, t_0→z_2로 표현되고, else 절을 위한 논리식은 e_0→z_1, z_1→z_2로 표현된다. 또한 p_0로부터 then절과 else절에서 배정을 당하는 변수로의 흐름을 나타내기 위해 p_0→z_0, p_0→z_1의 논리식으로 표현된다.

if 문에서 명시적 정보 누출이 발생할 때의 정보흐름을 논리식으로 나타내기 위해서는 if 문의 조건절이 참일 경우와 거짓일 경우를 나누어 고려해야 한다. if 문의 조건절이 참일 경우 then절의 정보흐름의 논리식은 (F↔z_1)∧(t_0↔z_2)로 표현되고, 거짓일 경우 else 절의 정보흐름의 논리식은 (e_0→z_1)∧(z_1→z_2)로 표현된다.

while 문의 경우는 반복될 때마다 변수간의 정보흐름 의존이 변한다. while 문의 몸체에서 변수간의 정보흐름 의존 관계를 정보 의존 그래프로 나타낸다. while 문을 충분한 횟수 k 만큼 반복한다면 변수간의 정보흐름 의존이 더 이상 변하지 않는 경우가 발생한다. 이때의 반복 횟수를 k라 했을 때 while 문의 정보흐름을 논리식으로 나타내기 위해서는 우선 while 문을 중첩된 if 문으로 변환한다. 변환된 if 문은 while 문을 k번 반복한 것과 동일한 의미를 갖는다. k는 실제 프로그램 실행 시 while 문을 반복하는 횟수보다 클 수 있다. 이렇게 k를 고려함으로써 정보의 흐름이 안전하지만 안전하지 않다고 거짓 경보를 줄 수 있다. 즉, 분석의 완전성(completeness)은 보장받지 못하지만 안전성(soundness)은 보장 받는다.

4. 명제 논리식 생성기

While 언어 구문에서 SSA 형태를 거쳐 [Fig. 5]에 정의된 명제 논리식 구문으로 변환 규칙에 따라 논리식을 생성한다. 논리식을 생성할 때 변수가 새로 생성될 때마다 변수를 구분하기 위해 인덱스가 붙도록 While 언어로부터 명제 논리식을 생성해야 한다. 이를 위해 변수로부터 현재 변수의 인덱스로의 사상을 I로서 정의한다.

$$I \in index = Var \rightarrow NonNegativeNumber$$

변환 규칙에서 I는 변수가 새로 정의될 때마다 변수에 새로운 인덱스를 붙이거나 기존에 정의된 변수를 사용할 때 인덱스를 붙이기 위해 사용된다. $I(x)$는 x의 현재 인덱스를 반환한다. $I[x \mapsto i]$는 I에서 변수 x의 인덱스를 i로 갱신한다는 의미이다. 논리식 변환을 위한 함수는 다음과 같다.

$$Check \subseteq 2^{Formula}$$
$$Flow \subseteq 2^{Formula}$$
$$\rightarrow_A : AExp \times Index \rightarrow Formula$$
$$\rightarrow_C : CExp \times Index \rightarrow Formula$$
$$\rightarrow_D : Dec \times Check \times Flow \times Index \rightarrow Check \times Flow \times Index$$
$$\rightarrow_S : Stmt \times Check \times Flow \times Index \rightarrow Stmt \times Check \times Flow \times Index$$
$$\rightarrow_S : Stmt \times Check \times Flow \times Index \rightarrow Check \times Flow \times Index$$

[Fig. 6] the function for translate to logic formula

산술식을 변환하기 위해서는 [Fig. 7]의 변환함수 $\rightarrow_A$를 사용한다. 변수는 현재의 인덱스를 첨자로 갖는 논리식 변수로, 상수는 낮은 보안 수준 L을 나타내는 진리 값 F로, 그리고 산술연산자가 포함된 산술식은 산술식의 피연산자 중 가장 높은 보안 수준이 산술식의 보안 수준을 정할 수 있도록 변환된다.

if 문이나 while 문에서 사용되는 조건 표현식은 [Fig. 7]의 변환함수 $\rightarrow_C$를 이용한다. 조건 표현식의 참, 거짓은 논리식의 T, F로 변환된다. 조건 표현식 중 관계연산자나 불리언 연산자가 포함된 표현식은 표현식을 나타낼 수 있는 논리식 변수로 변환된다. 예를 들어 x_0==1와 같은 조건 표현식은 $x_0$eq1와 같은 논리식 변수로 변환된다. 이 때 함수 α가 사용된다.

선언 부분의 변환은 [Fig. 8]의 변환함수 $\rightarrow_D$가 사용된다. 선언되는 변수 x의 보안수준에 따라 $x_0 \leftrightarrow$T나 $x_0 \leftrightarrow$F의 논리식이 생성되고, 제어와 정보흐름을 나타내는 ff 집합에 포함된다.

Arithmetic Expression

$$\frac{}{\langle x, I\rangle \rightarrow_A x_{I(x)}}\, x$$

$$\frac{}{\langle n, I\rangle \rightarrow_A L}\, n$$

$$\frac{\langle a_1, I\rangle \rightarrow_A f_1 \;\; \langle a_2, I\rangle \rightarrow_A f_2 \;\; f = f_1 \vee f_2}{\langle a_1 \; aop \; a_2, I\rangle \rightarrow_A f}\, arith\ op$$

Condition Expression

$$\frac{}{\langle true, I\rangle \rightarrow_C H}\, true$$

$$\frac{}{\langle false, I\rangle \rightarrow_C L}\, false$$

$$\frac{x_c = \alpha(a_1 \; rop \; a_2)}{\langle a_1 \; rop \; a_2, I\rangle \rightarrow_C x_c}\, rel\ op$$

$$\frac{x_c = \alpha(c_1 \; bop \; c_2)}{\langle c_1 \; bop \; c_2, I\rangle \rightarrow_C x_c}\, bool\ op$$

$$\frac{\langle c, I\rangle \rightarrow_C f}{\langle \neg c, I\rangle \rightarrow_C \neg f}\, neg$$

[Fig. 7] Transformation rules for arithmetic expressions and conditional expressions

문장 부분을 변환하기 전에 while 문을 if문으로 변환해야 한다. 변환은 while 문 중 가장 중첩된 안쪽의 while 문부터 바깥쪽의 while 문 순으로 변환한다. 변환 시 [Fig. 9]의 unroll 함수를 사용한다. unroll 함수에서 의존관계 그래프를 생성한다. 의존관계 그래프를 생성하면서 의존관계 그래프가 더 이상 변하지 않는 반복 횟수가 k일 때를 구한다. while 문이 k번 만큼 반복된 것과 같은 의미가 되도록 while문을 중첩된 if문으로 변환한다. 프로그램에 포함된 while 문을 모두 if 문으로 변환한 후 변환함수 $\rightarrow_S$를 적용하여 논리식을 생성한다.

배정문의 경우 정보흐름의 안전성 검사와 정보흐름을 모두 나타내는 논리식을 생성해야 한다. 안전성 검사를 위한 논리식 생성시 배정문의 lhs를 위해 현재 인덱스 i를 구하고, rhs를 위해 $\rightarrow_A$를 통해 산술식 a를 위한 논리식을 구하여 논리식 f_c를 생성하여 검사를 위한 논리식 집합 f_c에 포함시킨다. 정보흐름을 위한 논리식 생성시 lhs를 위해 현재 인덱스 i를 구하여 1 증가시키고, rhs를 위해 구한 논리식 f_c를 가지고 f_t를 생성하여 정보흐름을 위한 논리식 집합 ff에 포함시킨다. 문장의 연속 $S_1;S_2$에 대해서는 S_1이 변환되는 중간 과정을 나타내는 *comp1* 규칙, S_1이 다 변환된 경우를 위한 *comp2* 규칙을 활용하

Declaration

$$\frac{f = x_0 \leftrightarrow T \quad I' = I[x \mapsto 0]}{\langle high\ x := a, fc, ff, I\rangle \rightarrow_D \langle fc, \{f\} \cup ff, I'\rangle} dec_{high}$$

$$\frac{f = x_0 \leftrightarrow F \quad I' = I[x \mapsto 0]}{\langle low\ x := a, fc, ff, I\rangle \rightarrow_D \langle fc, \{f\} \cup ff, I'\rangle} dec_{low}$$

$$\frac{I' = I[x \mapsto 0]}{\langle unknown\ x := a, fc, ff, I\rangle \rightarrow_D \langle fc, ff, I'\rangle} dec_{unknown}$$

Statement

$$\frac{i = I(x) \quad \langle a, I\rangle \rightarrow_A f_a \quad f_c = f_a \rightarrow x_i \quad f_f = f_a \leftrightarrow x_{i+1} \quad I' = I[x \mapsto i+1]}{\langle x := a, fc, ff, I\rangle \rightarrow_S \langle \{f_c \cup fc, \{f_f\} \cup ff, I'\rangle} assign$$

$$\frac{\langle S_1, fc, ff, I\rangle \rightarrow_S \langle S_1', fc', ff', I'\rangle}{\langle S_1; S_2, fc, ff, I\rangle \rightarrow_S \langle S_1'; S_2, fc', ff', I'\rangle} comp1$$

$$\frac{\langle S_1, fc, ff, I\rangle \rightarrow_S \langle fc', ff', I'\rangle}{\langle S_1; S_2, fc, ff, I\rangle \rightarrow_S \langle S_2, fc', ff', I'\rangle} comp2$$

$$\frac{\begin{array}{cc} \langle fc'_{then}, fc'_{else}, ff'_{then}, ff'_{else}, I'\rangle = adjustDepth(I_{then}, I_{else}, fc_{then}, fc_{else}, ff_{then}, ff_{else}, I) & \\ \langle c, I\rangle \rightarrow_C f_{cond} & cc = \bigvee_{x \in FV(c)} (x_{I(x)} \leftrightarrow T) \\ fc_{impthen} = genImp(c, S_1, I, I') & \langle S_1, \phi, \phi, I\rangle \rightarrow_S \langle fc_{then}, ff_{then}, I_{then}\rangle \\ fc_{impelse} = gneImp(c, S_2, I, I') & \langle S_2, \phi, \phi, I\rangle \rightarrow_S \langle fc_{else}, ff_{else}, I_{else}\rangle \\ fc_{expthen} = (f_{cond} \wedge \neg cc) \rightarrow fc'_{then} & ff_{expthen} = (f_{cond} \wedge \neg cc) \rightarrow ff'_{then} \\ fc_{expelse} = (\neg f_{cond} \wedge \neg cc) \rightarrow fc'_{else} & ff_{expelse} = (\neg f_{cond} \wedge \neg cc) \rightarrow ff'_{else} \\ fc_{impif} = cc \rightarrow (fc'_{then} \wedge fc'_{else} \wedge fc_{impthen} \wedge fc_{impelse}) & ff_{impif} = cc \rightarrow (fc'_{then} \vee fc'_{else}) \end{array}}{\langle if\ c\ then\ S_1\ else\ S_2, fc, ff, I\rangle \rightarrow_S \langle fc_{expthen} \cup fc_{expelse} \cup fc_{impif} \cup fc, ff_{expthen} \cup ff_{expelse} \cup ff, I'\rangle} if$$

$$\frac{\langle a, I\rangle \rightarrow_A f_a \quad f_c = f_a \rightarrow F}{\langle print\ a, fc, ff, I\rangle \rightarrow_S \langle \{f_c\} \cup fc, ff, I\rangle} print$$

Subfunctions

$$conjunct(fs) = \bigwedge_{f \in elems(fs)} f \quad genImp(c, S, I, I') = \bigwedge_{\substack{x \in FV(c) \\ y_j \in LHS(S) \\ I(y) <= j < I'(y)}} (x_{I(x)} \rightarrow y_j)$$

$$genChecks(x, start, end) = \bigwedge_{i = start\ to\ end} (x_i \rightarrow x_i) \quad genEquals(x, start, end) = \bigwedge_{i = start\ to\ end} (x_i \leftrightarrow x_{i+1})$$

$$\begin{array}{l} adjustDepth(I_1, I_2, fc_1, fc_2, ff_1, ff_2, I) = \\ I' = \phi \\ foreach\ x \in (Dom(I_1) \cup Dom(I_2)) \{ \\ \begin{cases} fc_1 = fc_1 \cup (genChecks(x, I_1(x), I_2(x) - 1); & if\ I_1(x) < I_2(x) \\ ff_1 = ff_1 \cup (genEquals(x, I_1(x), I_2(x) - 1); & \\ I'[x \mapsto I_2(x)] & \\ fc_2 = fc_2 \cup (genChecks(x, I_2(x), I_1(x) - 1); & else\ if\ I_1(x) > I_2(x) \\ ff_2 = ff_2 \cup (genEquals(x, I_2(x), I_1(x) - 1); & \\ I'[x \mapsto I_1(x)] & \end{cases} \\ \} return\ (conjunct(fc_1), conjunct(fc_2), conjunct(ff_1), conjunct(ff_2), I') \end{array}$$

[Fig. 8] Transformation rules for statements and helper functions

여 논리식을 생성한다. if 문의 정보흐름 안전성 검사를 위해서는 묵시적 누출검사, 명시적 누출 검사 중 then 절이 실행될 때, 명시적 누출 검사 중 else 절이 실행될 때를 나누어 생성해야 한다. 우선 if 문의 조건절 c를 나타내는 논리식 f_{cond}와 명시적인지 묵시적인지 구분하기 위한 논리식 c_c를 생성한다. then 절과 else 절의 정보흐름

안전성을 각각 검사했을 때를 나타내는 fc_{then}, fc_{else}를 구한다. then 절에 대한 변환함수를 적용한 후와 else 절에 대한 변환함수를 적용한 후의 인덱스 맵 I_{then}과 I_{else}의 인덱스 깊이가 다를 수 있다. 여기서 인덱스 깊이가 같다는 것은 두 인덱스 맵에서 같은 변수의 인덱스는 같은 값을 갖는 경우를 의미한다. 예를 들어 모든 변수 x에 대해 I_{then}의 변수 x의 인덱스 $I_{then}(x)$와 I_{else}의 변수 x의 인덱스 $I_{else}(x)$가 같으면 I_{then}의 깊이와 I_{else}의 깊이가 같다고 한다. 인덱스 깊이를 같게 하기 위해 adjustDepth 함수를 사용한다. 인덱스 깊이가 같아진 fc'_{then}과 fc'_{else}를 이용해 if 문의 명시적 정보 누출을 나타내는 논리식을 생성할 수 있다.

doLoop function

$$
\begin{array}{l}
doLoop(s,g) \quad = \\
match\ s\ with \\
\mid skip \quad \longrightarrow \quad g \\
\mid print\ a \quad \longrightarrow \quad g \\
\mid x := a \quad \longrightarrow \quad g' = g \\
\qquad\qquad \cup \{z \rightsquigarrow x \mid z \rightsquigarrow y \in g \text{ and } y \in FV(a)\} \\
\qquad\qquad \cup \{y \rightsquigarrow x \mid y \in FV(a)\}; \\
\qquad\qquad g' \\
\mid S_1; S_2 \quad \longrightarrow \quad g' = doLoop(S_1, g); \\
\qquad\qquad g'' = doLoop(S_2, g'); \\
\qquad\qquad g'' \\
\mid if\ c\ then\ S_1\ else\ S_2 \quad \longrightarrow \quad g'_1 = doLoop(S_1, g); \\
\qquad\qquad g'_2 = doLoop(S_2, g); \\
\qquad\qquad (g'_1 \cap g'_2)
\end{array}
$$

depGraph function

$$
\begin{array}{l}
depGraph(s) = \\
g_{pre} = \phi; g_{cur} = \phi; k = 0; \\
do\{ \\
g_{cur} = doLoop(s, g_{pre}); \\
k = k + 1; \\
\}while(g_{pre} \neq g_{cur}); \\
k;
\end{array}
$$

unroll function

$$
\begin{array}{l}
unroll(while\ c\ do\ S) = \\
w = while\ c\ do\ S; \\
k = depGraph(S); \\
if\ k = 1\ then\ translate\ w\ into\ if\ c\ then\ S\ else\ skip \\
else\ if\ k = 2\ then\ translate\ w\ into\ if\ c\ then\ S\ else \\
(if\ c\ then\ S\ else\ skip) \\
\ldots
\end{array}
$$

[Fig. 9] function unroll to translate while statement into if statements

논리식 $fc_{expthen}$은 묵시적 누출이 발생하지 않고($\neg$cc) then절이 실행될 때(f_{cond}) fc'_{then}이 검사돼야 함을 나타낸다. 논리식 $fc_{expelse}$은 묵시적 누출이 발생하지 않고($\neg$cc) else절이 실행될 때($\neg f_{cond}$) fc'_{else}이와 같은 정보흐름이 나타날 수 있음을 나타낸다. 묵시적 정보누출 검사를 위한 논리식 fc_{impif}는 묵시적 누출이 발생하면(cc) then절이 검사되고 (fc'_{then}) else 절이 검사되며(fc'_{else}) then 절로의 묵시적 누출이 검사되고($fc_{impthen}$) else 절로의 묵시적 누출이 검사돼야($fc_{impelse}$) 한다는 것을 나타낸다.

5. While 언어 프로그램의 정보흐름분석

본 장에서는 예제 프로그램으로 정보흐름의 안전성 분석을 설명한다. [Fig. 3]의 프로그램을 SSA 형태의 프로그램으로 변환하면 [Fig. 4]와 같으며 명제 논리식 생성기인 SAT translator를 이용하여 SSA 형태의 프로그램을 [Fig. 10]와 같은 명제 논리식으로 변환한다.

1	h0↔T
2	l0↔F
3	
4	t0↔F
5	
6	z0↔F
7	(h0∨t0→l0)
8	¬(p0↔F)→(F→z1)∧(t0→z2)∧(e0→z1)∧(z1→z2)∧(p0→z0)∧(p0→z1)
9	(p0↔F)∧(p0e1)→(F→z1)∧(t0→z2)
10	(p0↔F)∧(¬p0e1)→(e0→z1)∧(z1→z2)
11	z2→F

[Fig. 10] Propositional logic formula of Fig.4

생성된 각 논리식의 문장을 s_n으로 나타날 때, ($s_1 \wedge s_2 \wedge \ldots s_n$)가 SAT 해결기에서 만족 가능할 경우 안전한 실행을 하나라도 갖고 있는 프로그램을 의미하며 안전한 프로그램이 되게 하는 프로그램 변수와 프로그램 실행 경로의 반례를 알 수 있다. 따라서 논리식 ($s_1 \wedge s_2 \wedge \ldots s_n$)의 만족가능성만으로 프로그램의 정보흐름이 안전하다는 것을 보장할 수 없다. 그러므로 분석에 유용한 결과를 얻기 위해서는 프로그램이 안전하지 않은지, 즉 $\neg$($s_1 \wedge s_2 \wedge \ldots s_n$)이 만족가능한지 판단함으로써 프로그램이 위험하다는 것을 판단할 수 있다.

[Fig. 10]의 논리식을 1행부터 11행까지 s_1 부터 s_{11}로 나타내고 명제 논리식을 $\neg$($s_1 \wedge s_2 \wedge \ldots s_{11}$)로 SAT 해결기를 통해 실행하였을 때, 정보흐름이 안전하면 아무 오류

메시지가 없으며 안전하지 않으면 <Table 2>와 같이 반례가 나타난다. 따라서 예제 프로그램은 정보흐름이 안전하지 않음을 나타낸다.

<Table 2> Counter-Examples

	h0	t0	l0	l1	p0	p0e1	e0	z0	z1	z2
s1	T	F	F	T	F	T	T	F	F	F
s2	T	F	F	T	F	T	F	F	F	F
s3	T	F	F	T	F	F	T	F	T	T
s4	T	F	F	T	F	F	F	F	F	F
s5	T	F	F	T	T	T	F	F	F	F

<Table 2>의 반례를 통해 어느 변수가 정보흐름에 안전하지 않은지 알 수 있다. s_1부터 s_4 까지 p_0가 F이므로 예제 프로그램 8행의 if 문에서 명시적 누출이 발생할 수 있음을 가리킨다. 그에 비해 s_5부터는 p_0가 T이므로 묵시적 누출이 발생함을 나타낸다. s1을 통해 어떤 실행에서 7행 문장이 h_0가 T이고 l_0가 F 이므로 안전하지 않다는 것을 알 수 있다. 하지만 다른 부분에서 안전하지 않은 정보흐름이 발생하는지는 알 수 없다. s_3의 경우 어떤 실행에서 7행 문장이 h_0가 T이고 l_0가 F 이므로 안전하지 않다는 것 뿐 만 아니라 p_0e_1이 F이므로 if 문의 else 절이 실행될 때 e_0가 T가 될 경우 z_0가 F이므로 else 절의 배정문 $z_1:=e_0$에 의해 if 문의 명시적 누출이 발생함을 알 수 있다.

기존 연구에 비해 안전한 프로그램을 안전하지 않다고 분석하는 경우는 없으며 어느 부분이 문제인지 반례를 통해 알 수 있고 다양한 프로그램 특성에 따라 설계가 쉽다.

6. 결론

스마트폰의 열풍으로 PC 환경뿐만 아니라 모바일 환경에서 인터넷을 사용하는 사람들이 매우 많아졌고 다양한 앱들을 통해 개인정보를 사용하고 있으나 프로그램의 안전성은 믿을 수 없는 상황이다.

본 논문은 SAT 해결기를 이용해서 프로그램 코드에서 정보흐름의 안전성을 검사하는 방법을 제안하였다. 안전성 검사를 위해 프로그램을 SSA 형태로 변환하고 SSA 형태의 프로그램을 명제 논리식으로 변환하였다. 명제 논리식의 구문 구조를 정의하였고 변환하기 위한 규칙을 제안하였다. 그 결과 SAT 해결기를 이용하여 변환된 명제 논리식으로 프로그램의 정보흐름이 안전한지 검사할 수 있었다. 정보흐름이 안전하지 않은 경우 안전하지 않은 반례를 생성하여 어느 부분의 정보흐름이 안전하지 않은지 확인할 수 있었다.

본 논문에 제안한 방법은 While 언어를 정의하여 예제 프로그램으로 설명하였다. 향후 연구에서는 안드로이드 환경에서 사용되는 앱의 정보흐름의 안전성을 분석할 수 있도록 연구를 수행할 것이다.

REFERENCES

[1] A. Sabelfeld and A. C. Myers, "Language-based information-flow security," IEEE J.Sel.A.Commun., vol. 21, no. 1 , pp. 5-19, Sep. 2006.

[2] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck, "Efficiently computing static single assignment form and the control dependence graph," ACM Trans. Program. Lang. Syst., vol. 13, no. 4 , pp. 451-490, Oct. 1991.

[3] Myung-Seong Yim, "Understanding the Factors that influence Website Retention and Privacy Unconcern After the Disclosure of Privacy Information," Journal of Digital convergence, The Korea Society of Digital Policy and Management, vol, 11, no 1, pp. 107-119, Jan. 2013.

[4] D. E. Denning, "A lattice model of secure information flow," Commun ACM, vol. 19, no. 5, pp. 236-243, May. 1976.

[5] D. E. Denning and P. J. Denning, "Certification of programs for secure information flow," Commun ACM, vol. 20, no. 7, pp. 504-513, Jul. 1977.

[6] K. G. Doh and S. C. Shin, "Detection of information leak by data flow analysis," SIGPLAN Not., vol. 37, no. 8, pp. 66-71, Aug. 2002.

[7] D. M. Volpano and G. Smith, "A Type-Based Approach to Program Security," in Proceedings of the 7th International Joint Conference CAAP/FASE on Theory and Practice of Software Development: Springer-Verlag, pp. 607-621, 1997.

[8] S. Hunt and D. Sands, "On flow-sensitive security types," SIGPLAN Not., vol. 41, no. 1, pp. 79-90, Jan. 2006.

[9] Y. Liu and A. Milanova, "Static analysis for inference of explicit information flow," in Proceedings of the 8th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, Atlanta, Georgia, pp. 50-56, 2008.

[10] R. Joshi and K. R. M. Leino, "A semantic approach to secure information flow," Science of Computer Programming, vol. 37, no. 1, pp. 113. 2000.

[11] T. Amtoft and A. Banerjee, "A logic for information flow analysis with an application to forward slicing of simple imperative programs," Sci.Comput.Program., vol. 64, no. 1, pp. 3-28, Jan. 2007.

[12] T. Amtoft and A. Banerjee, "Verification condition generation for conditional information flow," in Proceedings of the 2007 ACM workshop on Formal methods in security engineering, Fairfax, Virginia, USA, pp. 2-11, 2007.

[13] D'Silva, Vijay, Leopold Haller, and Daniel Kroening. "Satisfiability Solvers are Static Analysers," Eds. Antoine Miné and David Schmidt. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[14] Gergo. Barany, "Hybrid Information Flow Analysis for Programs with Array," Workshop on Verification and Program Transformation, 2016.

[15] Ono, Masahiro, et al. "SMART: A Propositional Logic-Based Trade Analysis and Risk Assessment Tool for a Complex Mission," Aerospace Conference, IEEE , pp. 1-15, 2015.

[16] Walch, Martin, Rouven Walter, and Wolfgang Küchlin. "Formal Analysis of the Linux Kernel Configuration with SAT Solving," in Proceedings of the 17th International Configuration Workshop. 2015.

[17] R. Sen and Y. N. Srikant. Executable analysis using abstract interpretation with circular linear progressions. In Proceedings of the Fifth IEEE/ACM International Conference on Formal Methods and Models for Codesign, pages 39 - 48. IEEE, 2007.

[18] Sik-Wan Cho, Won-Jun Jang, Hyung-Woo Lee, "Development of User Oriented Vulnerability Analysis Application on Smart Phone", Journal of the Korea Convergence Society, Vol. 3, No. 2, pp. 7-12, 2012.

[19] Seung-Soo Shin, "A Study on Multi-Media Contents Security Using Android Phone", Journal of the Korea Convergence Society, Vol. 3, No. 1, pp. 19-25, 2012.

김 제 민(Kim, Je Min)

- 2006년 2월 : 인하대학교(공학사)
- 2008년 2월 : 인하대학교(공학석사)
- 2008년 3월 ~ 현재 : 인하대학교 박사과정
- 관심분야 : 프로그램분석, 프로그래밍언어, 컴파일러 등
- E-Mail : jeminya@hanmail.net

고 훈 준(Kouh, Hoon Joon)

- 1998년 2월 : 인하대학교 (공학사)
- 2000년 2월 : 인하대학교 (공학석사)
- 2004년 2월 : 인하대학교 (공학박사)
- 2004년 3월 ~ 현재 : 경인여자대학교 교수
- 관심분야 : 프로그램분석, 뮤직테크놀러지, 교육프로그램 등
- E-Mail : hjkouh@kiwu.ac.kr