명령형 프로그램의 핵심부분에 대한 정보흐름 보안성의 데이타 흐름 분석

(Data Flow Analysis of Secure Information-Flow in Core Imperative Programs)

신 승 철 † 변 석 우 †† 정 주 희 ††† 도 경 구 ††††
(Seung Cheol Shin) (Sugwoo Byun) (Joohee Jeong) (Kyung-Goo Doh)

요 약 이 논문은 명령형 프로그램의 핵심 부분에 대한 정보흐름의 보안성을 데이타 흐름 분석법을 사용하여 예측하는 방법을 제시한다. 지금까지 제안된 분석 기법은 정보흐름이 안전한 프로그램을 안전하지 않다고 보수적으로 판정한다는 점에서 정밀도가 떨어지는 경우가 많이 있다. 이 논문에서는 이전의 구문 중심의 접근방법보다는 분석결과가 더 정밀한 새로운 분석법을 제안하고, 그 분석의 안전성을 증명한다.

키워드 : 정보흐름 보안, 정적분석, 데이타흐름분석

Abstract This paper uses the standard technique of data flow analysis to solve the problem of secure information-flow in core imperative programs. The existing methods tend to be overly conservative, giving "insecure" answers to many "secure" programs. The method described in this paper is designed to be more precise than previous syntactic approaches. The soundness of the analysis is proved.

Key words : Secure information-flow, static analysis, data flow analysis

1. 서 론

프로그램에서 정보 흐름(information flow)의 보안성(security)을 정적으로 결정하는 문제는 70년대 Denning 부부의 선구적인 논문[1,2]에서 제안된 이후 수 십 년 동안 연구되어 왔다. 정보흐름의 보안성이 보장되려면 불간섭 성질(noninterference property)이 성립해야 하는데, 이는 프로그램의 실행을 통해서 기밀 정보가 허가되지 않은 대상에 누출되지 않음을 의미한다.

이 문제를 구문적으로 푸는 방법으로는 데이타 흐름 분석(data flow analysis)을 통한 방법[3-5], 타입시스템(type systems)을 기반으로 하는 방법[6,7] 등이 있다. 이러한 방법들은 정보 흐름이 안전하지 않은(insecure) 프로그램에 대해서 "안전하다(secure)"고 틀린 분석결과를 절대로 내주지 않는다는 측면에서 대부분 분석의 안전함(soundness)이 증명되었다. 그러나 Denning 부부의 방법[2], Mizuno와 Schmidt의 접근방법[4], Volpano와 Smith의 타입시스템을 기반으로 하는 방법[6]은 모두 너무 보수적인 경향이 있어서, 정보흐름이 안전한 프로그램을 안전하지 않다고 판정하는 경우가 많이 있어서 분석의 정밀성이 떨어진다.

이 논문에서는 명령형 프로그램의 핵심 부분을 대상으로 데이타 흐름 분석의 표준 기법을 사용하여 정보흐름의 보안성을 예측하는 문제를 풀어본다. 이 논문에서 제시한 분석방법은 위에서 열거한 여러 보수적인 방법보다 더 정밀하도록 설계되었다. 그러나 제시한 분석방법은 구문 중심이기 때문에 본질적으로 Joshi과 Leino나 Sabelfeld와 Sands가 제시한 의미를 기반으로 하는 방법만큼 정밀할 수는 없다[8,9].

일반적으로 분석의 결과가 정밀해지면 질수록, 분석과 그 분석의 안전성 증명은 더 복잡해진다. 이 논문에서

† 정 회 원 : 동양대학교 컴퓨터공학부 교수
shin@dyu.ac.kr
†† 비 회 원 : 경성대학교 컴퓨터과학과 교수
swbyun@kyungsung.ac.kr
††† 비 회 원 : 경북대학교 수학교육과 교수
jhjeong@knu.ac.kr
†††† 비 회 원 : 한양대학교 전자컴퓨터공학부 교수
doh@cse.hanyang.ac.kr
Corresponding author
논문접수 : 2002년 11월 5일
심사완료 : 2004년 2월 5일

· 이 논문에서 "안전하다"라는 단어는 두 가지 의미로 쓰인다. 하나는 정보흐름이 안전하다(secure)는 의미이고, 다른 하나는 분석이 안전하다(sound)는 의미이다. 둘 다 우리말로 정착된 용어이므로 그대로 사용하였다. 단 독자들이 혼돈하지 않도록 문맥상 충분이 구분할 수 있게 하였음을 밝힌다.

분석을 명세한 데이타 흐름식은 일반적으로 많이 사용하는 *gen*과 *kill* 함수로 구성되어 있다. 따라서 추가적인 노력 없이도 표준 알고리즘으로 흐름식의 해를 구할 수 있고, 흐름식의 안전성도 일반적으로 많이 사용하는 표준적인 방법으로 증명할 수 있다.

이 논문의 구성은 다음과 같다. 2절에서는 간단한 예제를 통해서 정보 흐름의 보안성 문제를 약식으로 설명하고, 3절에서는 이 논문의 대상언어인 **While** 언어의 구문과 의미를 정의하고, 4절에서는 흐름그래프 구축 방법과 정보누출을 탐지하는 데이타 흐름식을 보여주고, 5절에서는 분석의 안전성을 증명하고, 6절에서 결론을 맺는다.

2. 정보흐름의 보안성

이 논문에서 다루는 명령형 프로그램에 대한 정보 흐름의 보안성 분석 문제를 다음과 같이 약식으로 설명할 수 있다. 변수가 비밀변수(높은 보안수준을 가진 변수)와 공개변수(낮은 보안수준을 가진 변수)로 구별되어 있는 프로그램에서, 공개변수의 최종 값을 보고 비밀변수에 초기에 저장된 값에 관련된 어떤 정보를 알아낼 수 있는지 여부를 검사하는 문제를 정보 흐름의 보안성 분석문제라고 한다. 즉, 공개변수의 실행전 값과 실행후 값을 보고 비밀변수의 실행전 값에 관한 정보를 절대로 알아낼 수 없는 경우, "정보 흐름은 안전이 보장된다."라고 한다.

예를 들어, x를 비밀변수, m과 n을 공개변수라고 할 때, 다음 저장문의 정보흐름은 안전하지 않다.

```
m := x
```

왜냐하면 변수 m에 저장된 값을 보고 x에 저장되어 있는 값을 알 수 있기 때문이다. 이러한 경우 변수 x에서 변수 m으로 명시적 정보 누출(explicit information leak)이 일어났다고 한다. 반면에 다음 저장문의 정보흐름은 안전하다고 한다.

```
x := m
```

왜냐하면 m 값을 알더라도 x에 저장된 값에 관련된 어떠한 정보도 알아낼 수 없기 때문이다. 다음 조건문을 생각해보자. 각 분기에 속한 문장만 따져보면 모두 정보흐름이 안전하지만, 조건문 전체적으로 보면 안전하다고 할 수 없다.

```
if x then m := 0 else m := 1
```

이 조건문에서 변수 x에 저장된 값이 변수 m에 그대로 저장되지는 않지만, 변수 m에 저장된 값을 보고 변수 x에 저장된 값에 대한 정보를 유추해낼 수 있다. 이러한 경우 변수 x에서 변수 m으로 묵시적 정보 누출(implicit information leak)이 일어났다고 한다. 이 두 가지 유형의 정보누출을 모두 이 논문에서는 즉시 누출(immediate leak)이라고 한다. 정보는 전이적으로 누출될 수도 있다. 예를 들어, 다음 프로그램에서

```
m := x;
n := m;
```

변수 x에 저장된 값은 변수 m 뿐 아니라 변수 n에 저장된 값을 보면 알 수 있다. 즉, 변수 x에서 변수 n으로의 정보 누출이 변수 m을 통하여 발생한다. 이러한 누출은 전이누출(transitive leak)이라고 한다.

누출된 정보는 복구되기도 한다. 예를 들어, 다음 프로그램의 정보흐름은 안전하다.

```
m := x;
m := n;
```

왜냐하면, 프로그램 실행이 끝난 후 변수 m에 변수 x의 값이 저장되어 있지 않기 때문이다. 이러한 상황을 "변수 x에서 변수 m으로의 정보 누출이 전방향으로 복구되었다(forwardly recovered)"라고 한다. 다음 프로그램도 안전하다.

```
x := m;
m := x;
```

왜냐하면, 프로그램 실행 전에 변수 x에 저장되어 있던 값은 변수 m에 저장되어 있던 공개값으로 변경되었기 때문이다. 이러한 상황을 "변수 x에서 변수 m으로의 정보 누출이 후방향으로 복구되었다(backwardly recovered)"라고 한다. 따라서 정보누출이 전혀 발생하지 않거나, 발생했더라도 모두 전방향 또는 후방향으로 복구된다면, 프로그램의 정보 흐름은 안전하다고 주장할 수 있다.

Denning 부부의 최초 방법[2], Mizuno와 Schmidt의 데이타 흐름 분석 알고리즘[4], Volpano와 Smith의 타입시스템을 기반으로 하는 방법[6]은 모두 위에서 언급한 정보 누출의 복구를 감지할 수 없고, 따라서 바로 위의 두 예제 프로그램에 대해서 정보흐름이 안전하지 않다고 판정한다는 점에서 분석의 정밀도가 낮다. 이 논문에서 소개하는 데이타 흐름 분석법은 마지막 두 예제 프로그램에 정보 누출이 없고, 따라서 정보흐름이 안전하다고 판정한다.

3. While 언어의 정의

간단한 핵심 명령형 언어 **While**의 구문구조(syntax)와 의미구조(semantics)를 정의해 보자. **While** 프로그램의 문장과 조건식을 인식하기 위해서 각 저장문(assignment), skip 문장, 조건식마다 유일한 번호 라벨을 부여한다.

구문 도메인(Syntax Domain):

$a \in$ **AExp** 산술식(arithmetic expressions)

$b \in$ **BExp** 논리식(boolean expressions)

$S \in$ **Stmt** 문장(statements)

$x \in$ **Var** 변수(variables)

$l \in$ **Lab** 라벨(labels)

구문 구조(Abstract Syntax):

$S ::= [x := a]^l \mid [\texttt{skip}]^l \mid S_1; S_2$

$\mid$ if $[b]^l$ then S_1 else S_2

$\mid$ while $[b]^l$ do S

구성(Configuration)과 변이규칙(Transition):

상태(state)는 다음과 같이 변수가 주어지면 변수에 저장되어 있는 값인 정수를 내주는 함수로 정의된다.

$$\sigma \in \textbf{State} = \textbf{Var} \rightarrow \textbf{Z}$$

의미구조의 구성은 $\langle S, \sigma \rangle$ 나 σ로 표시된다. 변이규칙은 주어진 프로그램의 실행이 끝나면 상태가 어떻게 변하는지 보여주는데 다음과 같은 형태로 표현된다.

$$\langle S, \sigma \rangle \rightarrow \sigma'$$

산술식과 논리식에 대한 의미 함수는 다음과 같이 이미 정의되어 있다고 가정한다.

$$\mathcal{A} : \textbf{AExp} \rightarrow (\textbf{State} \rightarrow \textbf{Z})$$

$$\mathcal{B} : \textbf{BExp} \rightarrow (\textbf{State} \rightarrow \textbf{T})$$

여기서 **Z**는 정수의 집합이고, **T**는 진리값의 집합이다.

의미구조(Natural Semantics):

[저장문] $\langle [x = a]^l, \sigma \rangle \rightarrow \sigma[x \mapsto \mathcal{A}[[a]]\sigma]$

[skip문] $\langle [\texttt{skip}]^l, \sigma \rangle \rightarrow \sigma$

[문장나열] $$\frac{\langle S_1, \sigma \rangle \rightarrow \sigma' \quad \langle S_2, \sigma' \rangle \rightarrow \sigma''}{\langle S_1; S_2, \sigma \rangle \rightarrow \sigma''}$$

[조건문#1] $$\frac{\langle S_1, \sigma \rangle \rightarrow \sigma'}{\langle \text{ if } [b]^l \text{ then } S_1 \text{ else } S_2, \sigma \rangle \rightarrow \sigma'} \quad \text{if } \mathcal{B}[[b]]\sigma = \mathit{true}$$

[조건문#2] $$\frac{\langle S_2, \sigma \rangle \rightarrow \sigma'}{\langle \text{ if } [b]^l \text{ then } S_1 \text{ else } S_2, \sigma \rangle \rightarrow \sigma'} \quad \text{if } \mathcal{B}[[b]]\sigma = \mathit{false}$$

[반복문#1] $$\frac{\langle S, \sigma \rangle \rightarrow \sigma' \quad \langle \text{ while } [b]^l \text{ do } S, \sigma' \rangle \rightarrow \sigma''}{\langle \text{ while } [b]^l \text{ do } S, \sigma \rangle \rightarrow \sigma''} \quad \text{if } \mathcal{B}[[b]]\sigma = \mathit{true}$$

[반복문#2] $\langle \texttt{while } [b]^l \texttt{ do } S, \sigma \rangle \rightarrow \sigma$ if $\mathcal{B}[[b]]\sigma = \mathit{false}$

4. 정보 누출 분석

이 절에서는 전통적인 데이타 흐름 분석기법을 사용하여 정보누출을 찾아내는 분석법을 살펴본다. 정보누출 분석을 위해서 **While** 프로그램을 흐름그래프(flow graph)변환한 후, 이 흐름그래프를 기초로 하여 정보누출을 탐지하는 데이타 흐름식을 제시한다.

4.1 흐름그래프

흐름그래프(flow graph)는 Nielson과 Nielson과 Hankin의 책에서 표현한 방법과 비슷하게 정의한다[10]. 흐름그래프에는 정상적인 제어흐름(control flow) 뿐만 아니라, 조건식 블록에서 조건문의 분기 또는 루프 몸체에 속해있는 각 문장 블록을 잇는 묵시적 흐름(implicit flow)도 포함되어 있다. 묵시적 흐름은 묵시적으로 일어나는 정보 누출을 감지하기 위해서 사용된다.

흐름그래프는 기본블록의 집합과 블록 사이를 잇는 (제어와 묵시적) 흐름의 집합으로 구성된다. 좀 더 정식으로 정의하면, **While** 문장 S의 흐름그래프는 다음과 같이 다섯 겹으로 구성된다.

$$\mathit{flowgraph}(S) = (\mathit{blocks}(S), \mathit{flow}(S), \mathit{flow}_I(S), \mathit{init}(S), \mathit{final}(S))$$

여기에서 각 함수는 아래와 같이 정의된다.

Blocks를 $[x := a]^l$ 또는 $[\texttt{skip}]^l$ 또는 $[b]^l$의 형태로 된 기본블록의 집합이라고 하자. 그러면 함수 *blocks*는 문장이 주어지면 기본 블록의 집합을 내주며, 다음과 같이 정의된다.

blocks: **Stmt** $\rightarrow \mathcal{P}$(**Blocks**)

$\mathit{blocks}([x := a]^l) = \{[x := a]^l\}$

$\mathit{blocks}([\texttt{skip}]^l) = \{[\texttt{skip}]^l\}$

$\mathit{blocks}(S_1; S_2) = \mathit{blocks}(S_1) \cup \mathit{blocks}(S_2)$

$\mathit{blocks}(\text{if } [b]^l \text{ then } S_1 \text{ else } S_2) = \{[b]^l\} \cup \mathit{blocks}(S_1) \cup \mathit{blocks}(S_2)$

$\mathit{blocks}(\text{while } [b]^l \text{ do } S) = \{[b]^l\} \cup \mathit{blocks}(S)$

흐름그래프에서 시작 블록은 항상 하나뿐이지만, 조건문 때문에 마지막 블록은 둘 이상이 될 수 있다. 함수 *init*는 문장에서 시작 블록의 라벨을 찾아주며, 다음과 같이 정의된다.

init **Stmt** $\rightarrow$ **Label**

$\mathit{init}([x := a]^l) = l$

$\mathit{init}([\texttt{skip}]^l) = l$

$\mathit{init}(S_1; S_2) = \mathit{init}(S_1)$

$\mathit{init}(\text{if } [b]^l \text{ then } S_1 \text{ else } S_2) = l$

$\mathit{init}(\text{while } [b]^l \text{ do } S) = l$

함수 *final*은 문장에서 마지막 블록의 라벨의 집합을 내주며, 다음과 같이 정의된다.

final : **Stmt** $\rightarrow \mathcal{P}$(**Lab**)

$\mathit{final}([x := a]^l) = \{l\}$

$\mathit{final}([\texttt{skip}]^l) = \{l\}$

$\mathit{final}(S_1; S_2) = \mathit{final}(S_2)$

$\mathit{final}(\text{if } [b]^l \text{ then } S_1 \text{ else } S_2) = \mathit{final}(S_1) \cup \mathit{final}(S_2)$

$\mathit{final}(\text{while } [b]^l \text{ do } S) = \{l\}$

함수 *flow*는 문장에 있는 블록들을 잇는 모든 제어흐름의 집합을 내주며, 다음과 같이 정의된다.

$$flow: \mathbf{Stmt} \rightarrow \mathcal{P}(\mathbf{Lab} \times \mathbf{Lab})$$
$$flow([x:=a]^l) = \emptyset$$
$$flow([\mathtt{skip}]^l) = \emptyset$$
$$flow(S_1;S_2) = flow(S_1) \cup flow(S_2) \cup \{(l, init(S_2)) \mid l \in final(S_1)\}$$
$$flow(\mathtt{if}\ [b]^l\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2) = flow(S_1) \cup flow(S_2) \cup \{(l, init(S_1)), (l, init(S_2))\}$$
$$flow(\mathtt{while}\ [b]^l\ \mathtt{do}\ S) = flow(S) \cup \{(l, init(S))\} \cup \{(l', l) \mid l' \in final(S)\}$$

다음 함수 $flow_I$는 문장에 있는 묵시적 흐름을 정의한다.

$$flow_I: \mathbf{Stmt} \rightarrow \mathcal{P}(\mathbf{Lab} \times \mathbf{Lab})$$
$$flow_I([x:=a]^l) = \emptyset$$
$$flow_I([\mathtt{skip}]^l) = \emptyset$$
$$flow_I(S_1;S_2) = flow_I(S_1) \cup flow_I(S_2)$$
$$flow_I(\mathtt{if}\ [b]^l\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2) = flow_I(S_1) \cup flow_I(S_2) \cup \{(l, l') \mid B^l \in blocks(S_1) \cup blocks(S_2)\}$$
$$flow_I(\mathtt{while}\ [b]^l\ \mathtt{do}\ S) = flow_I(S) \cup \{(l, l') \mid B^l \in blocks(S)\}$$

예를 들어 다음 프로그램을 보자.

$$S \equiv [z:=1]^1;\ \mathtt{while}\ [x>0]^2\ \mathtt{do}\ ([z:=z*y]^3; [x:=x-1]^4)$$

위 프로그램의 흐름그래프는 다음과 같다.

$$flowgraph(S) = (blocks(S), flow(S), flow_I(S), init(S), final(S))$$
$$= (\{[z:=1]^1, [x>0]^2, [z:=z*y]^3; [x:=x-1]^4\}, \{(1,2),(2,3),(3,4),(4,2)\}, \{(2,3),(2,4)\}, 1, \{2\})$$

4.2 분석

이 논문에서 정보의 보안 수준은 다음과 같다고 가정한다.

$$\mathbf{Sec} = \{\mathtt{Public}, \mathtt{Secret}\}$$
여기서 $\mathtt{Public} < \mathtt{Secret}$

여기서 $s<t$는 t가 s보다 보안수준이 절대적으로 높음을 의미한다. 프로그램에 있는 각 변수 x는 실행 전에 이미 보안수준이 정해져있다고 가정하고, 그 변수의 보안수준은 변수에 밑줄을 그어 $\underline{x}$와 같이 표시한다.

정보 누출은 $x \lhd y$로 표시되며, "변수 y에 저장되어 있는 정보는 변수 x로 누출된다"라고 한다. 여기서 변수 x를 누출된 변수(leaked variable)라고 한다. 분석이 진행되는 동안 프로그램의 정보누출은 다음과 같은 집합으로 수집된다.

$$L \subseteq \mathbf{Leak} = \{x \lhd y \mid x, y \in \mathbf{Var}, \underline{x} < \underline{y}\}$$

여기서 **Var**는 프로그램에 존재하는 모든 변수의 집합이다. 수집된 정보 누출은 명시적 누출이거나 묵시적 누출이고, 즉시 또는 전이 누출 중 하나이다. 정보 누출은 분석이 진행되는 동안 누출된 정보가 전방향 복구되면 L 집합에서 제거된다.

비밀변수(초기에 Secret으로 정해진 변수)에 저장된

$$gen: \mathbf{Blocks} \times \mathcal{P}(\mathbf{Leak}) \times \mathcal{P}(\mathbf{Lowered}) \times \mathcal{P}(\mathbf{Implicit}) \rightarrow \mathcal{P}(\mathbf{Leak}) \times \mathcal{P}(\mathbf{Lowered}) \times \mathcal{P}(\mathbf{Implicit})$$
$$gen([x:=a]^l, L, W, I) = (L', W', \emptyset)$$
$$\text{where } L' = \{x \lhd y \mid y \in FV(a), y \notin W, \underline{x} < \underline{y}\} \cup \{x \lhd y \mid z \in FV(a), z \lhd y \in L, \underline{x} < \underline{y}\} \cup \{x \lhd y \mid y \in I\}$$
$$W' = \{x \mid \underline{x} = \mathtt{Secret} \wedge (\forall v \in FV(a).(\forall w.v \lhd w \notin L) \vee v \in W) \wedge I = \emptyset\}$$
$$gen([\mathtt{skip}]^l, L, W, I) = (\emptyset, \emptyset, \emptyset)$$
$$gen([b]^l, L, W, I) = (\emptyset, \emptyset, I')$$
$$\text{where } I' = \{v \mid v \in FV(b), \underline{v} = \mathtt{Secret}, v \notin W\} \cup \{w \mid v \in FV(b), v \lhd w \in L\}$$

$$kill: \mathbf{Blocks} \times \mathcal{P}(\mathbf{Leak}) \times \mathcal{P}(\mathbf{Lowered}) \times \mathcal{P}(\mathbf{Implicit}) \rightarrow \mathcal{P}(\mathbf{Leak}) \times \mathcal{P}(\mathbf{Lowered}) \times \mathcal{P}(\mathbf{Implicit})$$
$$kill([x:=a]^l, L, W, I) = (\{x \lhd y \mid y \in \mathbf{Var}\}, \{x\}, \emptyset)$$
$$kill([\mathtt{skip}]^l, L, W, I) = (\emptyset, \emptyset, \emptyset)$$
$$kill([b]^l, L, W, I) = (\emptyset, \emptyset, \emptyset)$$

그림 1 *gen* 함수와 *kill* 함수

비밀정보가 공개정보로 변경되면 분석에서 그 변수는 낮아진 변수(lowered variable)로 분류된다. 이 변수는 다음과 같은 형태의 집합에 수집된다.

$W \subseteq$ **Lowered** = $\{x \mid x \in$ **Var**, $\underline{x}$ = Secret$\}$

낮아진 변수의 집합 W에 속해있는 변수 중에서 분석하는 도중 비밀 정보로 다시 변경되거나 비밀 정보의 영향을 받는다고 알려지면 그 집합에서 제거된다. 수집된 낮아진 변수들은 정보 누출이 후방향 복구되었는지를 알아내는데 사용한다.

조건식 블록(조건문이나 반복문)에 속한 비밀변수는 각각 해당 분기나 몸체에 묵시적으로 영향을 주므로 묵시적 누출을 찾는데 사용되며, 묵시 변수(implicit variable)로 분류되어 다음과 같은 형태의 집합에 수집된다.

$I \subseteq$ **Implicit** = $\{x \mid x \in$ **Var**, $\underline{x}$ = secret$\}$

그림 1의 *gen* 함수와 *kill* 함수는 흐름그래프 상의 각 블록의 출구에서 정보누출, 낮아진 변수, 묵시 변수들이 어떻게 생성되고 소멸되는지 정의한다. 이 함수에서 $FV(e)$는 표현식 e($\in$**AExp**$\cup$**BExp**)에 속한 변수들의 집합이다.

다음은 저장 블록인 $[x:=a]^l$에서 정보누출 $x \lhd y$가 생성되는 세가지 경우이다. (1) 즉시 누출(즉, 변수 y가 산술식 a에 속해있고, $\underline{x} \not< \underline{y}$ 가 참)이고, 후방향 복구가 되지 않았다(즉, 블록 l의 입구에서 y가 낮아진 변수로 분류되지 않았다). (2) 전이 누출(즉, 블록 l의 입구에서 $z \lhd y$가 누출로 분류되고, 변수 z가 산술식 a에 속해있고, $\underline{x} < \underline{y}$가 참)이다. (3) 묵시적 누출(즉, 블록 l의 입구에서 변수 y가 묵시 변수로 분류)이다.

다음 세 가지 조건이 모두 만족되면 저장 블록 $[x:=a]^l$에서 낮아진 변수 x가 생성된다. (1) 변수 x의 보안수준은 초기에 Secret으로 정해졌다. (2) 산술식 a에 속한 변수가 블록 l의 입구에서 누출 변수로 분류되지 않거나, 낮아진 변수로 분류된다. (3) 블록 l의 입구에서 묵시 변수가 없다.

저장 블록 $[x:=a]^l$에서는 x값이 수정되었기 때문에 이전에 생성되었던 x로의 모든 정보 누출은 *kill* 함수에 의해서 무조건 소멸되고, 낮아진 변수에서도 같은 이유로 x는 제거된다.

skip 블록에서는 아무 것도 생성되거나 소멸되지 않는다.

조건식 블록 $[b]^l$의 출구에서는 묵시 변수의 집합이 생성된다. 초기에 비밀변수이면서 블록 l의 입구에서 낮아진 변수로 분류되지 않은 논리식 b에 속한 변수는 묵시 변수이다. 변수 v가 논리식 b에 속해있고 블록 l의 입구에서 누출 $v \lhd w$이 존재하면, 블록 l의 출구에서 변수 w는 묵시 변수이다. 조건식 블록의 출구에서 생성된 묵시 변수 집합은 해당 조건 분기와 루프 몸체에 속한 각 저장 블록의 입구에서 참조될 뿐이다. 따라서 묵시 변수는 각각의 영역 범위 내에서 참조될 뿐이므로 소멸시킬 필요가 없다.

S를 분석 대상이 되는 프로그램, 즉, 최상위 문장이라고 한다.

$\mathbf{Leak}_{entry}, \mathbf{Leak}_{exit} : \mathbf{Lab} \to \mathcal{P}(\mathbf{Leak}) \times \mathcal{P}(\mathbf{Lowered}) \times \mathcal{P}(\mathbf{Implicit})$

$$\mathbf{Leak}_{entry}(l) = \begin{cases} (\emptyset, \emptyset, \emptyset) & \text{if } l = init(S) \\ (L, W, I) & otherwise \end{cases}$$

where $L = \cup \{L_i \mid (L_i, W_i, I_i) = \mathbf{Leak}_{exit}(l_i), (l_i, l) \in flow(S)\}$

$W = \cap \{W_i \mid (L_i, W_i, I_i) = \mathbf{Leak}_{exit}(l_i), (l_i, l) \in flow(S)\}$

$I = \cup \{I_i \mid (L_i, W_i, I_i) = \mathbf{Leak}_{exit}(l_i), (l_i, l) \in flow_I(S)\}$

$$\mathbf{Leak}_{exit}(l) = ((L \setminus L_{kill}) \cup L_{gen}, (W \setminus W_{kill}) \cup W_{gen}, (I \setminus I_{kill}) \cup I_{gen})$$

where $(L, W, I) = \mathbf{Leak}_{entry}(l)$

$(L_{kill}, W_{kill}, I_{kill}) = kill(B^l, L, W, I)$

$(L_{gen}, W_{gen}, I_{gen}) = gen(B^l, L, W, I)$

$B^l \in blocks(S)$

그림 2 정보 누출 분석을 위한 데이타 흐름식

분석 자체는 그림 2에서 정의한 대로 $\mathbf{Leak}_{entry}$와 $\mathbf{Leak}_{exit}$ 함수로 정의된다. $\mathbf{Leak}_{entry}(l)$은 블록 l의 입구로 들어오는 정보 누출 집합의 합집합, 낮아진 변수 집합의 교집합, 묵시 변수 집합의 합집합을 계산한다. $\mathbf{Leak}_{exit}(l)$은 (저장 블록의 경우) 블록 l의 출구에 이르기까지 수집한 정보누출의 집합과 낮아진 변수의 집합만들어 내고, (조건식 블록의 경우) 블록 l의 출구에서 묵시 변수의 집합을 만들어 낸다.

전방향 복구의 예로서 다음 **While** 프로그램을 생각해 보자.

$$[m:=x]^1;[m:=n]^2$$

여기서 $\underline{x}$ = Secret이고, $\underline{m}$ = $\underline{n}$ = Public이다.

$\mathbf{Leak}_{entry}(1) = (\emptyset, \emptyset, \emptyset)$

$\mathbf{Leak}_{exit}(1) = (\{m \lhd x\}, \emptyset, \emptyset)$

$\mathbf{Leak}_{entry}(2) = (\{m \lhd x\}, \emptyset, \emptyset)$

$\mathbf{Leak}_{exit}(2) = (\emptyset, \emptyset, \emptyset)$

블록 1의 출구에서 생성된 정보 누출은 기대한 대로 블록 2의 출구에서 복구된다. 따라서 이 프로그램에 정보 누출은 없다.

후방향 복구가 어떻게 이루어지는지 이해를 돕기 위해 예를 또 하나 보도록 하자.

$$[x:=m]^1;[m:=x]^2;[x:=y]^3;[m:=x]^4$$

여기서 $\underline{x}$ = $\underline{y}$ = Secret이고, $\underline{m}$ = Public이다.

$\mathbf{Leak}_{entry}(1) = (\emptyset, \emptyset, \emptyset)$

$\mathbf{Leak}_{exit}(1) = (\emptyset, \{x\}, \emptyset)$

$\mathbf{Leak}_{entry}(2) = (\emptyset, \{x\}, \emptyset)$

$\mathbf{Leak}_{exit}(2) = (\emptyset, \{x\}, \emptyset)$

$\mathbf{Leak}_{entry}(3) = (\emptyset, \{x\}, \emptyset)$

$\mathbf{Leak}_{exit}(3) = (\emptyset, \emptyset, \emptyset)$

$\mathbf{Leak}_{entry}(4) = (\emptyset, \emptyset, \emptyset)$

$\mathbf{Leak}_{exit}(4) = (\{m \lhd x\}, \emptyset, \emptyset)$

x는 블록 1의 출구에서 낮아진 변수가 되고, 따라서 정보 누출 $m \lhd x$는 블록 2의 출구에서 생성되지 않는다(후방향 복구). x는 블록 3의 출구에서부터 더 이상 낮아진 변수가 아니고, 따라서 정보 누출 $m \lhd x$는 블록 4의 출구에서 생성된다.

다음 예는 묵시적 누출이 어떻게 수집되는지 보여준다.

```
if [x>0]1 then [x:=m]2;[m:=0]3
          else [skip]4
```

여기서 $\underline{x}$ = Secret이고, $\underline{m}$ = Public이다.

$\mathbf{Leak}_{entry}(1) = (\emptyset, \emptyset, \emptyset)$

$\mathbf{Leak}_{exit}(1) = (\emptyset, \emptyset, \{x\})$

$\mathbf{Leak}_{entry}(2) = (\emptyset, \emptyset, \{x\})$

$\mathbf{Leak}_{exit}(2) = (\emptyset, \emptyset, \emptyset)$

$\mathbf{Leak}_{entry}(3) = (\emptyset, \emptyset, \{x\})$

$\mathbf{Leak}_{exit}(3) = (\{m \lhd x\}, \emptyset, \emptyset)$

$\mathbf{Leak}_{entry}(4) = (\emptyset, \emptyset, \{x\})$

$\mathbf{Leak}_{exit}(4) = (\emptyset, \emptyset, \emptyset)$

블록 2의 출구에서 x가 낮아진 변수가 되기는 하지만, 블록 3의 입구에서 x가 묵시 변수이므로 정보 누출 $m \lhd x$가 블록 3의 출구에서 생성된다.

4.3 데이타 흐름식의 해

그림 2와 같이 정의된 데이타 흐름식은 블록 개수에 따라서 연립 방정식의 형태를 이루고, 주어진 프로그램의 흐름 그래프가 사이클을 포함할 수 있기 때문에, 재귀적으로 정의될 수 있다. 데이타 흐름식이 항상 해(고정점)를 갖는다는 것을 보이기 위해서 데이타 흐름식의 정의역을 다음과 같이 부분 순서로 정의한다.

$\mathcal{D} = (\mathcal{P}(\mathbf{Leak}) \times \mathcal{P}(\mathbf{Lowered}) \times \mathcal{P}(\mathbf{Implicit}), \sqsubseteq)$

여기서 $L_1, L_2 \in \mathcal{P}(\mathbf{Leak})$, $W_1, W_2 \in \mathcal{P}(\mathbf{Lowered})$, $I_1, I_2 \in \mathcal{P}(\mathbf{Implicit})$ 일 때, $(L_1, W_1, I_1) \sqsubseteq (L_2, W_2, I_2)$ iff $L_1 \subseteq L_2$, $W_1 \supseteq W_2$, $I_1 \subseteq I_2$ 이다. 그리고 조인(join) 연산은 $(L_1, W_1, I_1) \sqcup (L_2, W_2, I_2) = (L_1 \cup L_2, W_1 \cap W_2, I_1 \cup I_2)$와 같이 정의된다. 그러면 정의역 $\mathcal{D}$의 $\top$은 (**Leak**,$\emptyset$,**Implicit**)이고 $\bot$은 ($\emptyset$,**Lowered**,$\emptyset$)이다. 정의역 $\mathcal{D}$는 유한하고 오름 사슬 조건(ascending chain condition)을 만족한다. 그러므로 $\mathcal{F}_l(\mathbf{Leak}_{entry}(l)) = \mathbf{Leak}_{exit}(l)$ 인 $\mathcal{D}$에 대한 전이 함수 $\mathcal{F}_l : \mathcal{D} \rightarrow \mathcal{D}$가 단조 함수이기만 하면 $\mathcal{F}_l$은 최소 고정점(least fixed point)를 갖는다. 즉, $(L_1, W_1, I_1) \sqsubseteq (L_2, W_2, I_2)$일때, $\mathcal{F}_l(L_1, W_1, I_1) \sqsubseteq \mathcal{F}_l(L_2, W_2, I_2)$임을, 다시 말해서

① $(L_1 \setminus L_{1kill}) \cup L_{1gen} \subseteq (L_2 \setminus L_{2kill}) \cup L_{2gen}$

② $(W_1 \setminus W_{1kill}) \cup W_{1gen} \supseteq (W_2 \setminus W_{2kill}) \cup W_{2gen}$

③ $(I_1 \setminus I_{1kill}) \cup I_{1gen} \subseteq (I_2 \setminus I_{2kill}) \cup I_{2gen}$

임을 보이면 된다. $[x := a]^l$의 경우, *gen*과 *kill* 함수의 정의로부터 $L_{1kill} = L_{2kill}$, $L_{1gen} \subseteq L_{2gen}$, $W_{1kill} = W_{2kill}$, $W_{1gen} \supseteq W_{2gen}$, $I_{1kill} = I_{2kill} = I_{1gen} = I_{2gen} = \emptyset$이므로 ①,②,③이 모두 성립함을 알 수 있다. $[b]^l$와 skip 블록의 경우도 쉽게 증명된다.

5. 분석의 안전성

분석의 본질 상 100% 정확한 답을 기대할 수 없다. 즉, 정보 흐름이 안전한 프로그램을 정부 누출이 있다고 부정확하게 판정할 수 있다. 그러나 정보흐름이 안전하지 않은 프로그램을 안전하다고 틀리게 판정하지는 말아야 한다. 이 절에서는 분석의 안전성(soundness)을 "주어진 프로그램의 분석하여 정보 누출이 없다고 판정되면, 그 프로그램의 정보 흐름은 안전하다고 보장된다"로 정의하고 이를 정식으로 증명한다.

증명은 분석이 불간섭(noninterference) 보안정책을 강제로 따른다는 사실을 보임으로써 수행한다. 프로그램에서 불간섭이란 비밀변수의 값이 다르다고 해도 공개변수의 값의 변화에 영향을 미치지 않는다는 뜻이다[9]. 따라서, 이 불간섭 성질을 만족하면, 비밀변수 값만 다른 두 프로그램을 각각 실행해보는 경우, 외부 침입자는 그 차이점을 결코 관찰할 수 없다[5].

각 블록의 입구(또는 출구)에서 공개값을 갖는 변수의 집합은 다음과 같이 정의한다.

정의 1: While 프로그램 S의 흐름 그래프에서, 블록 B^l의 입구와 출구에서 수집한 누출된 변수, 낮아진 변수, 묵시 변수를 각각 다음과 같이 이름 붙인다.

$(leak_e(l), lowered_e(l), implicit_e(l)) = \mathbf{Leak}_e(l)$

여기서 $e \in \{entry, exit\}$. 그러면 블록 B^l의 입구와 출구에서 공개값을 갖는 변수의 집합은 다음과 같이 정의된다.

$P_{l,entry}$ = B^l의 입구에서 공개값을 갖는 변수의 집합

$= \{x \mid x \in \mathbf{Var}, \underline{x} = \mathtt{Public}\}$

$\setminus \{x \mid x \lhd y \in leak_{entry}(l)\}$

$\setminus \{x \mid x \in implicit_{entry}(l)\}$

$\cup \{x \mid x \in lowered_{entry}(l)\}$

$P_{l,exit}$ = B^l의 출구에서 공개값을 갖는 변수의 집합

$= \{x \mid x \in \mathbf{Var}, \underline{x} = \mathtt{Public}\}$

$\setminus \{x \mid x \lhd y \in leak_{exit}(l)\}$

$\cup \{x \mid x \in lowered_{exit}(l)\}$ □

정의 2: (불간섭 성질) While 프로그램 S를 다음 조건

이 만족하는

$$\forall x \in \mathrm{P}_{init(S),entry}.\ \sigma_1(x) = \sigma_2(x)$$

서로 다른 임의의 σ_1과 σ_2를 가지고 각각 실행하여 다음과 같은 결과가 나오면,

$$\langle S, \sigma_1 \rangle \rightarrow \sigma_1'$$
$$\langle S, \sigma_2 \rangle \rightarrow \sigma_2'$$

다음이 성립한다.

$$\forall x \in \mathrm{P}_{init(S),entry}.\ \sigma_1'(x) = \sigma_2'(x) \qquad \square$$

While 프로그램 S가 이 불간섭 성질을 만족하면, 그 프로그램은 정보흐름의 안전이 보장된다(secure)라고 한다.

다음 정리 3에서는 임의의 프로그램 S의 실행이 종료한 후 공개값을 갖는 변수는 그 프로그램의 실행 전에 공개값을 갖는 변수에 의해서만 영향을 받는다고 주장한다.

정리 3: While 프로그램 S를 다음 조건이 만족하는

$$\forall x \in \mathrm{P}_{init(S),\ entry}.\ \sigma_1(x) = \sigma_2(x)$$

서로 다른 임의의 σ_1과 σ_2를 가지고 각각 실행하여 다음과 같은 결과가 나오면,

$$\langle S, \sigma_1 \rangle \rightarrow \sigma_1'$$
$$\langle S, \sigma_2 \rangle \rightarrow \sigma_2'$$

다음이 성립한다.

$$\forall x \in \bigcap_{l \in final(S)} \mathrm{P}_{l,exit}.\ \sigma_1'(x) = \sigma_2'(x)$$

증명:

변이가 이루어지는데 사용된 유추트리의 모양에 대한 귀납법으로 증명한다.

경우 [저장문]: $[x := a]^l$에 다음 조건을 만족하는

$$\forall v \in \mathrm{P}_{l,entry}.\ \sigma_1(v) = \sigma_2(v)$$

서로 다른 임의의 σ_1과 σ_2를 가지고, 의미규칙을 적용하면 다음과 같이 변이가 일어난다.

$$\langle [x := a]^l, \sigma_1 \rangle \rightarrow \sigma_1[x \mapsto \mathcal{A}[[a]]\sigma_1]$$
$$\langle [x := a]^l, \sigma_2 \rangle \rightarrow \sigma_2[x \mapsto \mathcal{A}[[a]]\sigma_2]$$

여기서 $\sigma_1[x \mapsto \mathcal{A}[[a]]\sigma_1]$은 σ_1'이라 하고, $\sigma_2[x \mapsto \mathcal{A}[[a]]\sigma_2]$은 σ_2'라고 하자. 그리고 다음이 성립함을 증명하면 된다.

$$\forall v \in \mathrm{P}_{l,exit}.\ \sigma_1'(v) = \sigma_2'(v)$$

여기서 σ_1과 σ_1'은 (그리고, σ_2과 σ_2'도) x의 값만 다르다. $x \notin \mathrm{P}_{l,exit}$인 경우에는 당연히 성립하므로, $x \in \mathrm{P}_{l,exit}$인 경우, $\sigma_1'(x) = \sigma_2'(x)$임만 보이면 된다. 그런데 $\sigma_1'(x) = \mathcal{A}[[a]]\sigma_1$이고, $\sigma_2'(x) = \mathcal{A}[[a]]\sigma_2$이므로, $\mathcal{A}[[a]]\sigma_1 = \mathcal{A}[[a]]\sigma_2$임을 증명하면 된다. $x \in \mathrm{P}_{l,exit}$이므로, $\mathrm{P}_{l,exit}$의 정의에 의해서, 다음 둘 중 하나가 성립함을 알 수 있다. ① $\underline{x}$ = Public이고, 모든 y에 대해서, $x \lhd y \notin leak_{exit}(l)$이다; ② $\underline{x}$ = Secret이고, $x \in lowered_{exit}(l)$이다. ①번의 경우, 그림 1과 그림 2의 정의에 의해서, 다음 식을 얻는다.

$$leak_{exit}(l) = leak_{entry}(l)$$
$$\setminus \{x \lhd y \mid y \in \mathbf{Var}\}$$
$$\cup \{x \lhd y \mid y \in FV(a), y \notin lowered_{entry}(l), \underline{x} < \underline{y}\}^{(i)}$$
$$\cup \{x \lhd y \mid z \in FV(a), z \lhd y \notin leak_{entry}(l), \underline{x} < \underline{y}\}^{(ii)}$$
$$\cup \{x \lhd y \mid y \in implicit_{entry}(l)\}^{(iii)}$$

그런데 $x \lhd y \notin leak_{exit}(l)$이기 때문에, 위의 (i), (ii), (iii)로 표시된 집합들은 모두 공집합일 수밖에 없다. 따라서 다음 세 논리식이 성립함을 유도할 수 있다.

$$\forall y \in FV(a).\ y \in lowered_{entry}(l) \vee \underline{y} = \text{Public}$$
$$\forall y \in FV(a).\ \forall w.\quad y \lhd w \notin leak_{entry}(l)$$
$$implicit_{entry}(l) = \emptyset$$

이제 여기서 논리적 암시를 통하여 유도하면, 다음 식을 얻는다.

$$\forall y \in FV(a).\ (\underline{y} = \text{Public} \wedge \forall w.\ y \lhd w \notin \leq ak_{entry}(l) \wedge y \notin implicit_{entry}(l)) \vee y \in lowered_{entry}(l)$$

그리고 이로부터 다음 식을 얻을 수 있다.

$$\forall y \in FV(a).\quad y \in \{v \mid \underline{v} = \text{Public}\} \setminus \{v \mid v \lhd w \in leak_{entry}(l)\} \setminus implicit_{entry}(l) \cup lowered_{entry}(l)$$

따라서 $\forall y \in FV(a).\ y \in \mathrm{P}_{l,entry}$가 성립함을 알 수 있다. 그런데, $\forall y \in FV(a).\ \sigma_1(y) = \sigma_2(y)$가 성립하므로, $\mathcal{A}[[a]]\sigma_1 = \mathcal{A}[[a]]\sigma_2$가 성립한다. ②번의 경우, $x \in lowered_{exit}(l)$은

$$(\forall v \in FV(a).\ (\underline{v} = \text{Public} \wedge \forall w.\ v \lhd w \notin leak_{entry}(l)) \vee v \in lowered_{entry}(l)) \wedge implicit_{entry}(l) = \emptyset$$

을 암시한다. 그러면 $\mathrm{P}_{l,entry}$의 정의에 의해서, $\forall v \in FV(a).\ v \in \mathrm{P}_{l,entry}$가 성립한다고 결론내릴 수 있다. 그래서 $\mathcal{A}[[a]]\sigma_1 = \mathcal{A}[[a]]\sigma_2$가 성립하고, 따라서 $\forall v \in \mathrm{P}_{l,entry}.\ \sigma_1'(v) = \sigma_2'(v)$가 성립한다.

경우 [skip문]: 사소하게 성립하므로 생략.

경우 [문장나열]: 다음을 만족하는 $S_1; S_2$에 대하여

$$\forall x \in \mathrm{P}_{init(S_1;S_2),\ entry}.\ \sigma_1(x) = \sigma_2(x) \qquad \textcircled{1}$$

서로 다른 임의의 σ_1과 σ_2를 가지고, 의미규칙을 적

용하여 다음과 같이 각각 변이가 일어난다고 가정하자.

$$\frac{\langle S_1,\ \sigma_1\rangle \rightarrow \sigma_1' \quad \langle S_2,\ \sigma_1'\rangle \rightarrow \sigma_1''}{\langle S_1;S_2,\ \sigma_1\rangle \rightarrow \sigma_1''}$$

$$\frac{\langle S_1,\ \sigma_2\rangle \rightarrow \sigma_2' \quad \langle S_2,\ \sigma_2'\rangle \rightarrow \sigma_2''}{\langle S_1;S_2,\ \sigma_2\rangle \rightarrow \sigma_2''}$$

이제 다음을 증명한다.

$$\forall x \in \bigcap_{l\in final(S_1;S_2)} P_{l,exit}\cdot \sigma_1''(x)=\sigma_2''(x) \qquad ㉡$$

귀납 가정에 의해서 $\langle S_1,\sigma_1\rangle \rightarrow \sigma_1'$와 $\langle S_1,\sigma_2\rangle \rightarrow \sigma_2'$일 때 $\forall x\in P_{init(S_1)\cdot entry}\cdot \sigma_1(x)=\sigma_2(x)$ 이면

$$\forall x \in \bigcap_{l\in final(S_1)} P_{l,exit}\cdot \sigma_1'(x)=\sigma_2'(x)$$

이고 또한

$\langle S_2,\sigma_1'\rangle \rightarrow \sigma_1''$ 와 $\langle S_2,\sigma_2'\rangle \rightarrow \sigma_2''$ 일 때

$$\forall x\in P_{init(S_2),entry}\cdot \sigma_1'(x)=\sigma_2'(x)$$

이면

$$\forall x \in \bigcap_{l\in final(S_2)} P_{l,exit}\cdot \sigma_1''(x)=\sigma_2''(x)$$

이다. 이때

$init(S_1;S_2)=init(S_1)$, $final(S_1;S_2)=final(S_2)$ 이고

$$\bigcap_{l\in final(S_1)} P_{l,exit} = P_{init(S_2),entry}$$

이므로 ㉠으로부터 ㉡이 성립함을 알 수 있다.

경우 [조건문]: 표현을 간략하게 하기 위해서 if $[b]^l$ then S_1 else S_2 $=cond([b]^l,S_1,S_2)$라고 놓고, 다음 조건을 만족하는

$$\forall x\in \mathbf{P}_{init(cond([b]^l,S_1,S_2)),entry}\cdot \sigma_1(x)=\sigma_2(x) \qquad ㉢$$

서로 다른 임의의 σ_1과 σ_2에 대하여, b의 연산 결과에 따라 각 경우로 나누어서 증명한다.

(가) $\mathcal{B}[[b]]\sigma_1 = \mathcal{B}[[b]]\sigma_2 = true$일 때

다음의 변이가 각각 일어난다고 가정하자.

$$\frac{\langle S_1,\ \sigma_1\rangle \rightarrow \sigma_1'}{\langle \text{ if } [b]^k \text{ then } S_1 \text{ else } S_2,\ \sigma_1\rangle \rightarrow \sigma_1'}$$

$$\frac{\langle S_1,\ \sigma_2\rangle \rightarrow \sigma_2'}{\langle \text{ if } [b]^k \text{ then } S_1 \text{ else } S_2,\ \sigma_2\rangle \rightarrow \sigma_2'}$$

그리고 귀납 가정에 의해서

$$\forall x\in \mathbf{P}_{init(S_1),entry}\cdot \sigma_1'(x)=\sigma_2'(x) \qquad ㉣$$

이면

$$\forall x \in \bigcap_{l\in final(S_1)} \mathbf{P}_{l,exit}\cdot \sigma_1'(x)=\sigma_2'(x) \qquad ㉤$$

이다.

그런데 $\mathbf{P}_{init(cond([b]^k,S_1,S_2)),entry} = \mathbf{P}_{init(S_1),entry}$ 이므로 ㉢이면 ㉣임을 알 수 있다. 따라서 ㉤이고 또한

$$\bigcap_{l\in final(S_1)} \mathbf{P}_{l,exit} \supseteq \bigcap_{l\in final(cond([b]^k,S_1,S_2))} \mathbf{P}_{l,exit}$$

이므로

$$\forall x \in \bigcap_{l\in final(cond([b]^k,S_1,S_2))} P_{l,exit}\cdot \sigma_1'(x)=\sigma_2'(x)$$

임을 알 수 있다.

(나) $\mathcal{B}[[b]]\sigma_1 = \mathcal{B}[[b]]\sigma_2 = false$일 때

(가)와 같은 방법으로 쉽게 증명된다.

(다) $\mathcal{B}[[b]]\sigma_1 = true$이고 $\mathcal{B}[[b]]\sigma_2 = false$일 때

각각 다음의 변이가 일어난다고 가정할 수 있다.

$$\frac{\langle S_1,\ \sigma_1\rangle \rightarrow \sigma_1'}{\langle \text{ if } [b]^k \text{ then } S_1 \text{ else } S_2,\ \sigma_1\rangle \rightarrow \sigma_1'}$$

$$\frac{\langle S_2,\ \sigma_2\rangle \rightarrow \sigma_2'}{\langle \text{ if } [b]^k \text{ then } S_1 \text{ else } S_2,\ \sigma_2\rangle \rightarrow \sigma_2'}$$

k의 입구에서 σ_1과 σ_2의 공개변수의 값이 모두 같다고 가정하였는데, $\mathcal{B}[[b]]\sigma_1 \neq \mathcal{B}[[b]]\sigma_2$의 경우에는 b에 비밀변수가 반드시 존재함을 의미한다. 즉,

$$FV(b) - \mathbf{P}_{init(cond([b]^k,S_1,S_2))} \neq \emptyset$$

이다. 따라서 S_1과 S_2의 모든 블록들이 $[b]^k$의 비밀변수로부터 묵시적인 정보흐름을 통하여 영향을 받게 되므로 gen의 정의에 의해서 S_1과 S_2에서 생성되는 낮아진 변수 집합 W는 모두 $\emptyset$이 될 수 밖에 없다. 그러므로

$$\bigcap_{l\in final(S_1)} \mathbf{P}_{l,exit} \subseteq \mathbf{P}_{init(S_1),entry}$$

$$\bigcap_{l\in final(S_2)} \mathbf{P}_{l,exit} \subseteq \mathbf{P}_{init(S_2),entry}$$

이다. 여기서 (가)와 (나)에서와 마찬가지로

$$\mathbf{P}_{init(cond([b]^k,S_1,S_2)),entry} = \mathbf{P}_{init(S_1),entry} (= \mathbf{P}_{init(S_2),entry})$$

이다. 그리고

$$final(cond([b]^k,S_1,S_2)) = final(S_1)\cup final(S_2)$$

에 의해서

$$\bigcap_{l\in final(S_1)} \mathbf{P}_{l,exit} \cap \bigcap_{l\in final(S_2)} \mathbf{P}_{l,exit} = \bigcap_{l\in final(cond([b]^k,S_1;S_2))} \mathbf{P}_{l,exit}$$

이므로 ㉢으로부터

$$\forall x \in \bigcap_{l\in final(cond([b]^k,S_1;S_2))} \mathbf{P}_{l,exit}\cdot \sigma_1'(x)=\sigma_2'(x)$$

임을 알 수 있다.

(라) $\mathcal{B}[[b]]\sigma_1 = false$이고 $\mathcal{B}[[b]]\sigma_2 = true$일 때 (다)와 같은 방법으로 증명된다.

경우 [반복문]: 표현을 간략하게 하기 위하여

while $[b]^k$ do $S = loop([b]^k,S)$ 라고 놓자.

다음 조건을 만족하는 임의의 σ_1과 σ_2에 대하여

$$\forall x\in \mathbf{P}_{init(loop([b]^k,S)),entry}\cdot \sigma_1(x)=\sigma_2(x) \qquad ㉥$$

b의 연산 결과에 따른 각 경우에 다음이 성립함을 증명한다.

$$\forall x \in \bigcap_{l\in final(loop([b]^k,S))} \mathbf{P}_{l,exit}\cdot \sigma_1'(x)=\sigma_2'(x)$$

(마) $\mathcal{B}[[b]]\sigma_1 = \mathcal{B}[[b]]\sigma_2 = true$일 때

다음을 가정하자.

$$\frac{\langle S,\sigma_1\rangle \rightarrow \sigma_1'' \quad \langle \text{ while [b] }^k \text{ do } S,\sigma_1''\rangle \rightarrow \sigma_1'}{\langle \text{ while [} b\text{] }^k \text{ do } S,\sigma_1\rangle \rightarrow \sigma_1'}$$

$$\frac{\langle S,\sigma_2\rangle \rightarrow \sigma_2'' \quad \langle \text{ while [} b\text{] }^k \text{ do } S,\sigma_2''\rangle \rightarrow \sigma_2'}{\langle \text{ while [} b\text{] }^k \text{ do } S,\sigma_2\rangle \rightarrow \sigma_2'}$$

그러면 귀납 가정에 의해서

$$\forall x \in \mathbf{P}_{init(S),entry} \cdot \sigma_1(x) = \sigma_2(x) \quad ㉮$$

이면

$$\forall x \in \bigcap_{l \in final(S)} \mathrm{P}_{l,exit} \cdot \sigma_1''(x) = \sigma_2''(x) \quad ㉯$$

이고 또

$$\forall x \in \mathbf{P}_{init(loop([b]^k,S)),entry} \cdot \sigma_1''(x) = \sigma_2''(x) \quad ㉰$$

이면

$$\forall x \in \bigcap_{l \in final(loop([b]^k,S)),exit} \mathbf{P}_{l,exit} \cdot \sigma_1'(x) = \sigma_2'(x) \quad ㉱$$

이다. 이때

$$\mathbf{P}_{init(loop([b]^k,S)),entry} = \mathbf{P}_{init(S),entry}$$

이므로 ㉲으로부터 ㉮을 얻고 그래서 ㉯임을 알 수 있다. 또한 그림 2와 정의 1에 의해서

$$\mathbf{P}_{init(loop([b]^k,S)),entry} \subseteq \bigcap_{l \in final(loop([b]^k,S))} \mathbf{P}_{l,exit}$$

이므로 ㉯으로부터 ㉰을 얻고 귀납 가정으로 ㉱임을 알 수 있다.

(바) $\mathcal{B}[[b]]\sigma_1 = \mathcal{B}[[b]]\sigma_2 = false$일 때

사소하게 성립한다.

(사) $\mathcal{B}[[b]]\sigma_1 = true$이고 $\mathcal{B}[[b]]\sigma_2 = false$일 때

다음을 가정하자.

$$\frac{\langle S,\sigma_1\rangle \rightarrow \sigma_1'' \quad \langle \text{ while [} b\text{] }^k \text{ do } S,\sigma_1''\rangle \rightarrow \sigma_1'}{\langle \text{ while [} b\text{] }^k \text{ do } S,\sigma_1\rangle \rightarrow \sigma_1'}$$

$$\langle \text{ while [} b\text{] }^k \text{ do } S,\sigma_2\rangle \rightarrow \sigma_2' \quad (\text{여기서 } \sigma_2 = \sigma_2')$$

그러면 $FV(b)$에는 반드시 비밀 값을 가진 변수가 들어있다. 또한 (다)의 경우와 같은 이유로

$$\mathbf{P}_{init(loop([b]^k,S)),entry} \supseteq \bigcap_{l \in final(loop([b]^t,S))} \mathbf{P}_{l,exit}$$

이고

$$\bigcap_{l \in final(loop([b]^t,S))} \mathbf{P}_{l,exit}$$

에 속한 변수들은 S 안에서 절대로 갱신되지 않는다. 따라서

$$\forall x \in \bigcap_{l \in final(loop([b]^t,S))} \mathbf{P}_{l,exit}\, \sigma_1'(x) = \sigma_2'(x)$$

임을 알 수 있다.

(아) $\mathcal{B}[[b]]\sigma_1 = false$이고 $\mathcal{B}[[b]]\sigma_2 = true$일 때

(사)와 같은 방법으로 증명된다. □

다음 정리 4에서는 분석결과 프로그램에서 정보 누출을 발견하지 못했다면, 그 프로그램은 정보누출이 절대 발생하지 않는다고 주장한다. 즉, 그 프로그램은 불간섭 성질을 만족한다고 보장한다는 것이다.

정리 4: (분석의 안전성) 종료하는 모든 **While** 프로그램 S에 대해서, $l \in final(S)$인 모든 l에 대해서 분석결과로 $leak_{exit}(l) = \emptyset$가 나온다면, S는 불간섭성질을 만족하면서 정보흐름이 안전하다.

증명: S를 다음 조건이 만족하는

$$\forall x \in \mathbf{P}_{init(S),entry} \cdot \sigma_1(x) = \sigma_2(x)$$

서로 다른 임의의 σ_1과 σ_2를 가지고 각각 실행하여 다음과 같은 결과가 나오면,

$$\langle S,\sigma_1\rangle \rightarrow \sigma_1'$$

$$\langle S,\sigma_2\rangle \rightarrow \sigma_2'$$

다음이 성립함을 증명하면 된다.

$$\forall x \in \mathbf{P}_{init(S),entry} \cdot \sigma_1'(x) = \sigma_2'(x) \quad ㉳$$

그런데 정리 3에 의해서 다음이 성립함을 알 수 있다.

$l \in final(S)$인 모든 l에 대해서

$$\forall x \in \mathbf{P}_{l,exit} \cdot \sigma_1'(x) = \sigma_2'(x)$$

게다가 $l \in final(S)$인 모든 l에 대해서 $\preceq ak_{exit}(l) = \emptyset$이므로, 다음 식을 유추할 수 있다.

$$\mathbf{P}_{init(S),entry} \subseteq \mathbf{P}_{l,exit}$$

따라서 ㉳이 성립한다. □

6. 결 론

이 논문에서는 데이타 흐름 분석을 통하여 정보 흐름의 안전성 문제를 푸는 해법을 제시하였다. 이 논문에서 제안한 분석법은 재래적인 구문기반 방법과 비교해서 더 정밀하다. 그리고 "분석을 통하여 프로그램에 정보누출이 없다고 판정되면, 그 프로그램의 정보흐름은 안전하다고 보장된다"는 분석의 안전성을 증명하였다.

참 고 문 헌

[1] D.E. Denning, A lattice model of secure information flow, *Communication of the ACM*, 19(5): 236-243, 1976.

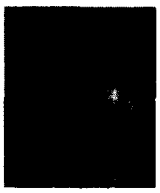
[2] D.E. Denning and P.J. Denning, Certification of programs for secure information flow, *Communication of the ACM*, 20(7):504-512, 1977.

[3] G.R. Andrews and R.P. Reitman, An axiomatic approach to information flow in programs, ACM Transactions on Programming Languages and Systems, 2(1):56-76, 1980.

[4] M. Mizuno and D.A. Schmidt, A security flow control algorithm and its denotational semantics correctness proof, *Formal Aspects of Computing*, 4:722-754, 1992.

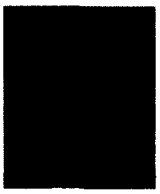
[5] J.-P. Banatre, C. Bryce, and D. Metayer, Compile-time detection of information flow in sequential programs, In D. Gollmann, editor, *Computer Security - ESORICS'94, the 3rd European Symposium on Research in Computer Security, Lecture Notes in Computer Science*, volume 875,

pages 55-73, Springer-Verlag, 1994.
[6] D. Volpano and G. Smith, A type-based approach to program security, In *TAPSOFT'97, the 7th International Conference on Theory and Practice of Software Development, Lecture Notes in Computer Science*, pages 607-621, Springer-Verlag, 1997.
[7] D. Volpano, G. Smith and C. Irvine, A sound type system for secure information flow, *Journal of Computer Security*, 4:1-21, 2001.
[8] R. Joshi and K.R.M. Leino, A semantic approach to secure information flow, *Science of Computer Programming*, 37:113-138, 2000.
[9] A. Sabelfeld and D. Sands, A PER model of secure information flow in sequential programs, *Higher-Order and Symbolic Computations*, 14:59-91, 2001.
[10] F. Nielson, H.R. Nielson and C. Hankin, *Principles of Program Analysis*, Springer, 1999.
[11] A. Sabelfeld and A.C. Myers, Language-based information-flow security, *IEEE Journal on Selected Areas in Communications*, To appear, 2002.
[12] J.A. Goguen and J. Meseguer, Unwinding and inference control, In Proc. *IEEE Symposium on Security and Privacy*, pages 75-86, 1984.



신 승 철

1987년 인하대학교 전자계산학과(학사) 1989년 인하대학교 전자계산학과(석사) 1996년 인하대학교 전자계산학과(박사) 1999년~2000년 Kansas State University 연구원. 1996년~현재 동양대학교 컴퓨터학부 조교수. 관심분야는 프로그래밍언어, 프로그램 분석 및 검증, 소프트웨어 보안, 이론 전산학(수리논리)



변 석 우

1976년~1980년 숭실대학교 전자계산(학사). 1980년~1982년 숭실대학교 전자계산(석사). 1982년~1999년 ETRI 책임연구원. 1988년~1994년 영국 University of East Anglia 전산학(박사). 1998년~현재 경성대학교 정보과학부 조교수. 관심분야는 rewriting system, 함수형 프로그래밍, 의미론, 정형 시스템 등

정 주 희

1979년 서울대학교 요업공학과(학사). 1981년 KAIST 재료공학과(석사). 1991년 UC Berkeley Dept. of Math.(Ph.D.) 1997년~경북대학교 수학교육과 조교수 관심분야는 수리논리학, 보편대수학, 의미론

도 경 구

1980년 한양대학교 산업공학(학사). 1987년 미국 Iowa State University 전산학(석사). 1992년 미국 Kansas State University 전산학(박사). 1993년 4월~1995년 9월 일본 University of Aizu 교수. 1995년 9월~현재 한양대학교 부교수. 관심분야는 프로그래밍 언어, 프로그램 분석 및 검증, 소프트웨어 보안