요약해석을 이용한 정보흐름 제어

신승철[1] 도경구[2] 이수호[1]
[1]동양대학교 컴퓨터공학부 [2]한양대학교 컴퓨터공학과
{scshin, hoya}@phenix.dyu.ac.kr, doh@cse.hanyang.ac.kr

Information Flow Control using Abstract Interpretation

Seung Cheol Shin[1], Kyung-Goo Doh[2], Suho Lee[1]
[1]Dongyang University, [2]Hanyang University

요 약

정보 흐름 보안성은 프로그램 보안성 검증 문제의 하나로서 주어진 프로그램이 각 변수들에 미리 정해진 보안수준이 허용하는 범위 내에서만 정보가 교환되는지를 검증하는 것이다. 프로그램 정적 분석을 위한 여러 가지 방법들 중에서 요약 해석법은 프로그램 분석의 기초를 마련할 뿐 아니라 저장기반 데이터흐름 분석 문제를 간략하게 해주는 장점을 가지고 있다. 본 논문에서는 요약해석을 이용하여 정보흐름 보안성을 검증하는 방법을 설명하고 이를 이용하여 프로그램 개발 시에 유용한 디버깅 정보를 제공하는 방법을 제시한다.

1. 서 론

컴퓨터와 인터넷 사용의 급증으로 컴퓨터 통신뿐 아니라 소프트웨어의 보안성이 매우 중요한 문제가 되고 있다. 소프트웨어 또는 프로그램 보안성 문제는 특정 프로그래밍 언어가 가지는 보안의 취약점을 검사하는 문제와 함께 작성된 프로그램이 보안과 관련된 특정 명세를 만족하는지를 검사하는 프로그램 검증 문제를 포함한다. 본 논문은 프로그램 보안성 문제 중에서 정보 흐름의 안전성/보안성(secure information flow) 문제를 다루고자 한다.

주어진 프로그램에 나타나는 변수들을 비밀변수와 공개변수로 구분하고, 비밀변수에 저장되어 있는 정보는 보호해야 하고 공개변수에 저장되어 있는 정보는 외부로 공개해도 된다고 생각해보자. 그러면 프로그램 실행 도중 비밀변수의 값이 공개변수에 저장이 되어 궁극적으로 외부에 누출되지 않도록 프로그램을 작성해야 할 것이다. 이와 같이 보안성을 만족하는 프로그램을 작성하기는 쉽지 않으므로 작성된 프로그램에 대하여 이를 확인하는 것이 필요하다.

비밀변수로부터 공개변수로의 정보 누출이 없는 프로그램을 "정보흐름이 안전한 프로그램"이라고 한다[1,2]. 주어진 프로그램에 대하여 비밀변수와 공개변수 사이의 정보 누출이 없는지를 검사하기 위해 지금까지 데이터흐름분석[3], 요약 해석[4,5], 타입시스템[6] 등을 이용하는 방법이 연구되어 왔고, 최근에는 의미구조를 이용한 방법[7]도 발표되고 있으며 데이터흐름분석과 SMV[8]와 같은 모델검사[9,10]의 관계[11]를 이용하는 방법[12]도 시도되었다.

본 논문은 간단한 명령형 언어 **While** 프로그램에 대한 정보흐름의 안전성을 검증하기 위하여 요약 해석을 이용하는 방법을 설명하고, 이를 확장하여 프로그램 개발 시에 유용한 디버깅 정보를 제공하는 방법을 제시한다. 2절에서 정보흐름 보안성을 정의하고 3절에서 While 프로그램의 문법과 의미구조를 보여주고, 4절과 5절에서 요약해석을 이용하여 정보흐름 보안성을 검증하는 방법을 설명한 후에 6절에서 이를 확장하여 디버깅 정보를 얻는 방법을 제시한다.

* 이 논문은 2000년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2000-003-E00250)

2. 정보흐름의 보안성

본 논문에서 다루는 정보흐름의 보안성은 명령형 프로그램에서 변수들간의 종속관계를 통해 발생하는 변수들간의 영향을 관찰하여 의도하지 않은 영향이 발생하는 지를 검사하는 것이다. 주어진 프로그램의 변수들은 모두 초기에 보안수준이 정해지는데 높은 보안 수준의 변수에 저장된 정보가 낮은 보안 수준의 변수로 누출되지 않아야 한다. 이것을 불간섭(noninterference)이라고 한다.

변수들에 부여될 수 있는 보안 수준들의 집합은 Sec = {public, secret}으로 나타내고, 이것은 부분순서 (Sec, $\sqsubseteq$)를 형성하며 public$\sqsubset$secret을 만족한다. 각 변수의 보안 수준 초기 값을 내주는 함수 $sec \in Var \to Sec$를 이용하면 변수 x와 y의 보안 수준에 대한 *join* 연산은 다음과 같이 정의될 수 있다. $sec(x) \sqcup sec(y) = \sqcup\{sec(x), sec(y)\} = sec(y)$ *iff* $sec(x) \sqsubseteq sec(y)$.

주어진 프로그램에서 발생 가능한 정보의 누출은 $a := x$와 같은 배정문에서 $sec(a) \sqsubset sec(x)$ 일 때 명시적으로 (explicit leak) 일어나거나, if x then $a := 0$ else $a := 1$와 같은 선택문에서 $sec(a) \sqsubset sec(x)$ 일 때 묵시적으로 (implicit leak) 발생한다.

정보의 누출은 전이적(transitive; indirect)으로도 발생할 수 있는데 이미 다른 변수로부터 누출된 정보를 가지고 있는 변수가 다른 변수에 그 정보를 재누출하는 경우를 말한다. 예를 들어, $sec(a) \sqsubseteq sec(b) \sqsubset sec(y)$ 일 때 프로그램 ($a := y$; $b := a$;)와 같이 y의 정보가 a를 통하여 b로 누출된다.

본 논문에서는 프로그램의 모든 변수의 보안 수준이 초기값에 비해 최종값이 커지지 않았을 경우에 프로그램의 정보흐름이 안전하다고 할 수 있다는 사실을 이용하여 프로그램의 안전성을 검증한다.

3. 대상 프로그램의 의미구조

본 논문에서 정보 흐름 보안성을 검증하는 대상이 되는 프로그램은 간단한 명령형 언어인 **While** 언어의 프로그램이며 다음의 문법 구조를 가지고 있다.

$$S ::= x := e \mid \text{if } e \text{ then } S \text{ else } S \mid \text{while } e \text{ do } S \mid S ; S \mid \text{skip}$$

$$e ::= c \mid x \mid e \; op \; e$$

여기서 c, x, op는 각각 상수, 변수, 이진 연산자이며, e는 표현식, S는 명령문을 나타낸다.

While 프로그램의 의미구조는 동작적 의미구조 표시방법(operational semantics)의 하나인 자연적 의미구조 표시방법(natural semantics)으로 다음과 같다.

$$\langle x := e, \sigma\rangle \rightarrow \sigma[[e]/x]$$

$$\frac{\langle s_1, \sigma\rangle \rightarrow \sigma'}{\langle \text{if } e \text{ then } s_1 \text{ else } s_2, \sigma\rangle \rightarrow \sigma'} \text{ if } [e]\sigma = \text{true}$$

$$\frac{\langle s_2, \sigma\rangle \rightarrow \sigma'}{\langle \text{if } e \text{ then } s_1 \text{ else } s_2, \sigma\rangle \rightarrow \sigma'} \text{ if } [e]\sigma = \text{false}$$

$$\frac{\langle s, \sigma\rangle \rightarrow \sigma' \quad \langle \text{while } e \text{ do } s, \sigma'\rangle \rightarrow \sigma''}{\langle \text{while } e \text{ do } s, \sigma\rangle \rightarrow \sigma''} \text{ if } [e]\sigma = \text{true}$$

$$\langle \text{while } e \text{ do } s, \sigma\rangle \rightarrow \sigma \text{ if } [e]\sigma = \text{false}$$

$$\frac{\langle s_1, \sigma\rangle \rightarrow \sigma' \quad \langle s_2, \sigma'\rangle \rightarrow \sigma''}{\langle s_1; s_2, \sigma\rangle \rightarrow \sigma''}$$

$$\langle \text{skip}, \sigma\rangle \rightarrow \sigma$$

여기서 프로그램의 의미는 $\langle S, \sigma\rangle \rightarrow \sigma'$와 같이 전이로 표현하는데, 이는 상태함수 σ가 주어지고 명령문 S를 실행하면 결과 상태함수가 σ'이 됨을 의미한다. 여기서 $\sigma \in$ State = Var→Val, Val = Int + Bool, $[e]\sigma$는 상태함수 σ하에서 표현식 e를 계산한 결과를 나타내고, $\sigma[v/x]$는 상태함수 σ에서 변수 x를 값 v로 갱신함을 나타낸다.

4. 정보 흐름의 요약 해석

앞 절에서 보여준 **While** 프로그램의 의미구조는 실제 값(concrete value)을 가지고 프로그램의 구체적인 실행(concrete interpretation)을 표현하고 있는 반면, 요약 해석(abstract interpretation)은 요약된 값(abstract value)을 가지고 요약적인 실행을 한다고 할 수 있다. 이 논문에서 다루는 정보 흐름의 요약 해석에서 상태는 두 가지 면에서 구체적인 실행과 다르다.

첫째는 구체 정의역이 아닌 요약 정의역의 값들을 다루는데 정보흐름 제어에서 이용되는 요약 정의역(abstract domain)은 2절에서 설명한 보안수준들의 집합 Sec이다. 프로그램을 요약 해석하는 도중 변수들의 보안수준은 $\sigma^{\#}$: Var → Sec$_\perp$ 와 같은 요약 상태함수를 통해 유지된다. Sec$_\perp$는 $\sigma^{\#}$이 부분함수임을 나타내는데 $\perp_{Sec}$는 해당 변수의 보안수준이 정의되어 있지 않음(undefined)을 나타낸다. 상태함수 $\sigma_1^{\#}$와 $\sigma_2^{\#}$사이의 join 연산 $\sigma_1^{\#} \sqcup \sigma_2^{\#}$도 각 변수들의 보안수준의 join에 의해 정의된다. 즉, $(\sigma_1^{\#} \sqcup \sigma_2^{\#})(x) = \sigma_1^{\#}(x) \sqcup \sigma_2^{\#}(x)$이다.

둘째로, 조건문이나 반복문의 조건식에 포함되어 있어서 본체에 묵시적인 영향을 주는 변수들의 집합은 따로 모아서 본체에 전달해 주어야 하므로, 상태함수와는 별도로 이 집합을 유지해 주어야 한다. 따라서 요약해석은 $\langle S, A, \sigma^{\#}\rangle \rightarrow \sigma^{\#'}$와 같이 전이로 표현하는데, 이는 묵시적으로 영향을 주는 변수의 집합 A와 상태함수 $\sigma^{\#}$가 주어지고 명령문 S를 요약해석하면 결과 상태함수가 $\sigma^{\#'}$이 됨을 의미한다. **While** 프로그램의 요약해석은 다음과 같이 정의된다. 여기서 $V(e)$는 수식 e에 나타난 변수들의 집합이다.

$$\langle x := e, A, \sigma^{\#}\rangle \rightarrow \sigma^{\#}[sc/x]$$
$$\text{where } sc = \sqcup\{\sigma^{\#}(y) \mid y \in A \cup V(e)\}$$

$$\frac{\langle S_1, A \cup V(e), \sigma^{\#}\rangle \rightarrow \sigma^{\#'} \quad \langle S_2, A \cup V(e), \sigma^{\#}\rangle \rightarrow \sigma^{\#''}}{\langle \text{if } e \text{ then } S_1 \text{ else } S_2, A, \sigma^{\#}\rangle \rightarrow \sigma^{\#'} \sqcup \sigma^{\#''}}$$

$$\frac{\langle S, A \cup V(e), \sigma^{\#}\rangle \rightarrow \sigma^{\#'}}{\langle \text{while } e \text{ do } S, A, \sigma^{\#}\rangle \rightarrow \sigma^{\#'}}$$

$$\frac{\langle S_1, A, \sigma^{\#}\rangle \rightarrow \sigma^{\#'} \quad \langle S_2, A, \sigma^{\#'}\rangle \rightarrow \sigma^{\#''}}{\langle S_1; S_2, A, \sigma^{\#}\rangle \rightarrow \sigma^{\#''}}$$

$$\langle \text{skip}, A, \sigma^{\#}\rangle \rightarrow \sigma^{\#}$$

5. 정보 흐름의 보안성 검증

여기서는 앞 절에서 설명된 요약 해석의 결과를 이용하여 주어진 프로그램의 정보 흐름 보안성을 검증하는 방법을 설명한다. 2절에서 설명된 바와 같이 프로그램 실행 전에 프로그램의 각 변수들에 대하여 보안수준이 정해진다. 이를 보안수준 초기 함수 $sec \in \sigma^{\#}$를 통해 나타내자. 그러면 초기 상태는 $\langle S, \varnothing, sec\rangle$이다. 주어진 프로그램은 S이고 이 프로그램 실행에 묵시적으로 영향을 주는 변수 집합은 공집합이고 변수 환경은 sec로 초기화되었다. 그렇다면 주어진 프로그램 S의 정보흐름 보안성은 다음과 같이 검증된다.

[정리] (정보 흐름 보안성)

$\langle S, \varnothing, sec\rangle \rightarrow \sigma^{\#}$ and $\sigma^{\#} \sqsubseteq$ sec *if and only if* **While** 프로그램 S는 정보흐름에 있어 정보 누출이 없는 안전한 프로그램이다.

우리는 위 정리를 증명하고 있으며 이의 구현도 진행중이다. 이러한 결과를 통해 주어진 프로그램의 요약 해석으로 얻어진 결과를 바탕으로 변수들의 초기값과 최종값을 비교하여 프로그램의 정보 흐름 보안성을 검증할 수 있다.

6. 정보 누출 정보의 생성

앞 절에서 보여준 요약 해석으로는 주어진 프로그램이 정보 누출의 가능성이 있는지의 여부를 알아낼 수 있을 뿐만 아니라, 요약 해석 후에 $sec(x) \sqsubset \sigma^{\#}(x)$가 되는 x를 찾음으로써 어떤 변수가 누출된 정보를 가지게 되는 지를 알

아낼 수 있다. 그러나 비록 x가 누출된 정보를 가질 수 있음을 안다하더라도 어느 변수로부터 그 정보가 누출되는지 또는 프로그램의 어느 위치에서 그 정보 누출이 일어날 수 있는지는 알 수가 없다. 이러한 자세한 것들을 컴파일 시간에 알 수 있다면 프로그램의 디버깅 중에 매우 유용하게 사용할 수 있을 것이다.

이 절에서는 4절에서 정의된 요약 해석을 디버깅에 유용한 정보들을 제공하도록 확장해본다. 먼저 변수들의 보안수준 상태함수 $\sigma^{\#}$: Var → $Sec_{\perp}$ 과 더불어 변수들의 정보누출 상태함수 σ^{+}: Var → *Powerset*(Var×Trace)를 추가한다. 이때, Label = {p | p는 주어진 프로그램의 포인트 또는 레이블} 이고, Trace = Label* 이다. 즉, Trace는 주어진 프로그램의 레이블들로 이루어진 모든 문자열들의 집합이다. 여기서 유의할 것은 프로그램 포인트는 skip과 배정문에만 주어진다는 것이다.

따라서 확장된 요약해석은 $\langle S, A, \sigma^{\#}, \sigma^{+}\rangle \rightarrow \langle \sigma^{\#'}, \sigma^{+'}\rangle$ 와 같이 전이로 표현하는데, 이는 누출에 관련된 변수 및 위치 정보가 전달됨을 의미한다. σ^{+}의 초기값은 모든 변수 x에 대하여 $\sigma^{+}(x) = \varnothing$인 함수가 된다. 이는, 초기에는 모든 변수가 누출된 정보를 가지지 않음을 나타낸다. σ^{+}가 어떻게 갱신되고 유지되는지는 다음의 확장된 요약 해석으로 보여준다. 사실 여기에 나타나는 프로그램들은 레이블이 붙은 프로그램이다. 이것에 대해 따로 설명하지 않고 다음의 요약 해석으로 대신한다.

$$\langle p : x := e, A, \sigma^{\#}, \sigma^{+}\rangle \rightarrow \langle \sigma^{\#}[sc/x], \sigma^{+}[tr/x]\rangle$$

where $sc = \sqcup\{\sigma^{\#}(y) \mid y \in A \cup V(e)\}$

and $tr = \{(v, p \cdot t) \mid (v, t) \in \sigma^{+}(y), \sigma^{\#}(y) = sc, y \in A \cup V(e)\}$
$\cup \{(y, p) \mid \sigma^{+}(y) = \varnothing, \sigma^{\#}(y) = sc, y \in A \cup V(e)\}$

$$\frac{\langle S_1, A \cup V(e), \sigma^{\#}, \sigma^{+}\rangle \rightarrow \langle \sigma^{\#'}, \sigma^{+'}\rangle \quad \langle S_2, A \cup V(e), \sigma^{\#}, \sigma^{+}\rangle \rightarrow \langle \sigma^{\#''}, \sigma^{+''}\rangle}{\langle \text{if } e \text{ then } S_1 \text{ else } S_2, A, \sigma^{\#}, \sigma^{+}\rangle \rightarrow \langle \sigma^{\#'} \sqcup \sigma^{\#''}, \sigma^{+'} \sqcup \sigma^{+''}\rangle}$$

$$\frac{\langle S, A \cup V(e), \sigma^{\#}, \sigma^{+}\rangle \rightarrow \langle \sigma^{\#'}, \sigma^{+'}\rangle}{\langle \text{while } e \text{ do } S, A, \sigma^{\#}, \sigma^{+}\rangle \rightarrow \langle \sigma^{\#'}, \sigma^{+'}\rangle}$$

$$\frac{\langle S_1, A, \sigma^{\#}, \sigma^{+}\rangle \rightarrow \langle \sigma^{\#'}, \sigma^{+'}\rangle \quad \langle S_2, A, \sigma^{\#'}, \sigma^{+'}\rangle \rightarrow \langle \sigma^{\#''}, \sigma^{+''}\rangle}{\langle S_1; S_2, A, \sigma^{\#}, \sigma^{+}\rangle \rightarrow \langle \sigma^{\#''}, \sigma^{+''}\rangle}$$

$$\langle p : \text{skip}, A, \sigma^{\#}, \sigma^{+}\rangle \rightarrow \langle \sigma^{\#}, \sigma^{+}\rangle$$

배정문에 대한 규칙에서 $p \cdot t$는 프로그램 레이블 p와 트레이스 tr이 접합됨을 나타낸다. 이 규칙의 의미는 변수 x에 영향을 주는 변수들 중에서 보안수준이 가장 높은 변수의 보안수준이 변수 x의 보안수준으로 갱신되고 또한 x의 보안수준을 배정하게 된 경위를 나타내는 프로그램 트레이스 tr이 만들어져서 새로운 정보누출 환경 $\sigma^{+'}$을 만들게 된다.

7. 결론

본 논문은 명령형 언어 **While** 프로그램에 대하여 정보흐름 안전성을 검증하기 위하여 요약 해석을 이용하는 방법을 설명하고 이를 확장하여 프로그램 개발시에 유용한 디버깅 정보를 제공하는 방법을 제시하였다.

참고문헌

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

[2] D.E. Denning and P.J. Denning, Certification of programs for secure information flow, Communications of ACM, 20(7), pp.504-513, 1977.

[3] J.-P.Banatre, C.Bryce, D. Le Metayer, Compile-time detection of information flow in sequential programs, Proc. European Symposium on Research in Computer Security, Lecture Notes in Computer Science, vol.875, pp.55-73, 1994.

[4] M. Mizuno and D. Schmidt, A security flow control algorithm and its denotational semantics-based correctness proof, Formal Aspects of Computing, 4, 727-754, 1992.

[5] S.Kuninobu et al, An efficient information flow analysis of recursive programs based on a lattice model of security classes, ICICS 2001, Lecture Notes in Computer Science, vol.2229, pp.292-303, 2001

[6] D. Volpano, G. Smith, C. Irvine, A sound type system for secure flow analysis, Journal of Computer Security 4(3), pp.1-21, 1996.

[7] R. Joshi and K.R.M. Leino, A semantic approach to secure information flow, Science of Computer Programming, 37, pp.113-138, 2000.

[8] K. L. McMillan, The SMV system for SMV version 2.5.4

[9] E.M. Clarke, O. Grumberg and D.E. Long, Model checking and abstraction, ACM Transactions on Programming Languages and Systems, 16(5):1512-1542, 1994.

[10] M. M. Muller-Olm, D. Schmidt and B. Steffen, "Model-checking: A tutorial introduction", In G. File and A. Cortesi, editors, Proc. of the 6th Static Analysis Symposium, Lecture Notes in Computer Science, Springer, 1999.

[11] D.A. Schmit, Data flow analysis is a model checking of abstract interpretation, In Proc. of the 25th ACM Symp. on Principles of Programming Languages, ACM Press, 1998.

[12] K.-G. Doh and S.C. Shin, "Analysis of secure information flow by model checking", In Proc. of the 2nd Asian Workshop on Programming Languages and Systems, pp.255-236, 2001.