논•문 16-3-5

# 바이트코드 정보흐름 분석기의 구현 (An Implementation of Information-Flow Analyzer for Bytecode)

조 순 희, 신 승 철
동양대학교 컴퓨터공학부
{chosh036,scshin}@phenix.dyu.ac.kr
도 경 구[1]
한양대학교 컴퓨터공학과
doh@cse.hanyang.ac.kr

## 요 약

프로그램은 실행을 통해서 작성자가 알아채지 못하는 사이에 정보의 누출이 발생할 수 있는데, 정보의 누출이 없는 프로그램을 정보흐름이 안전한 프로그램이라고 한다. 본 논문에서는 프로그램에서의 정보누출 여부를 검증하기 위해, 실험대상이 되는 While 프로그램을 바이트코드 JVML0로 바꾸고, 이것을 요약해석을 통하여 프로그램의 정보흐름이 안전한 지를 검사한다.

## 1. 서론

인터넷상에서 개인의 정보를 공개해야만 다운받을 수 있거나 실행되는 프로그램이 있다고 가정해 보자. 이 프로그램을 받기 위해 사용자는 개인의 정보를 입력하게 되고 이 과정에서 정보의 누출이 발생할 수 있다.

프로그램에서도 마찬가지이다. 변수의 정보가 공개되어도 좋은 변수를 공개변수라 하고, 변수의 정보가 공개되어서는 아니 되는 변수를 비밀변수라 하자. 작성자가 알아채지 못하는 사이 비밀변수의 정보가 공개변수로 넘어가게 된다면 이는 정보누출이 발생한 것이다.

이렇게 정보누출의 여부를 검사하기 위해

1 본 연구는 한국과학재단 목적기초연구(과제번호:R01-2000-00287) 지원으로 수행되었음.

서 우리는 각 변수마다 보안 수준(security level)을 부여하고 낮은 수준의 변수로 높은 수준의 변수의 정보가 흘러들어 갔는지 여부를 알아내려고 한다.

프로그램의 정보누출을 정적으로 검증하는 기법은 1970년대 Denning에 의해 연구되기 시작했다[1,2]. 검증기법에는 타입체계를 기반으로 하는 방법[7], 자료흐름분석을 기반으로 하는 방법[4,11,12], 요약해석을 이용하는 방법[6,10,13], 의미구조를 이용하여 접근하는 방법[8,9] 등이 있다. 본 논문에서는 요약해석을 이용하여 프로그램의 안전성을 검증하는 방법을 사용한다.

먼저 2절에서는 우리가 사용할 바이트코드(Bytecode) 언어인 JVML0에 대해서 소개하고, 3절에서는 실험을 위해 사용할 While 언어와 While을 JVML0로 변환하기 위한 규칙들을 설명하고, 4절에서 이를 가지고 요약해석을 하는 방법을 설명하고, 5절에서는 While 프로그램을 가지고 한 실험 결과를, 마지막으로 6절에서는 결론에 대해서 설명한다.

## 2. 프로그램의 안전성

프로그램의 정보흐름이 안전하다는 것은 보안수준이 높은 변수의 정보가 보안수준이 낮은 변수로 누출되지 않는다는 뜻이다. 본 논문에서는 주어진 바이트코드 프로그램에서 정보흐름의 안전성을 검사하는 방법을 다룬다.

프로그램 내의 변수들은 모두 초기에 보안수준이 정해져 있다고 가정한다. 초기에 부여된 변수들의 보안수준의 집합을 $Sec$ = {public, secret}으로 나타내고, 여기서 public은 공개되어도 좋은 공개변수의 보안수준을 나타내고, secret은 공개되지 말아야 하는 비밀변수의 보안수준을 나타낸다. 그리고 이들 사이는 부분순서 ($Sec$, $<$)가 형성되며, public $<$ secret의 순서를 만족하며, 이는 public의 보안수준보다 secret의 보안수준이 높음을 의미한다.

보안수준이 낮은 변수의 처음 값과 끝 값이 보안수준이 높은 변수의 처음 값으로부터 영향을 받지 않을 때 "두 변수 사이에 불간섭관계(noninterference)가 있다"라고 하며, 이 관계를 만족하는 프로그램을 정보흐름이 안전한 프로그램이라고 한다[3].

바이트코드에서 정보누출이 발생하는 경우를 살펴보자. 변수 x는 비밀변수이고, m과 n은 공개변수라고 가정하고 아래의 예를 보자.

```
1  load x
2  store m
```

여기에서 변수 x의 정보가 변수 m에 저장됨으로 인해서 실행 후 m을 관찰하면 비밀변수 x의 정보를 얻을 수 있으므로 위의 프로그램은 안전한 프로그램이 아니다. 이런 경우의 정보누출을 명시적 정보누출(explicit leak)이라고 한다. 또 다른 예를 보자.

```
1  push 1
2  load x
3  eq
4  if 8
5  push 0
6  store m
7  goto 10
8  push 1
9  store m
```

위의 프로그램은 변수 x의 값에 따라 m에 저장되는 값이 다르다. 이 경우도 m의 정보를 관찰함으로써 x의 정보를 유추할 수 있어 정보누출이 일어났다고 할 수 있고, 이러한 정보누출을 묵시적 정보누출(implicit leak)이라고 한다. 또 다른 예를 보자.

```
1  load x
2  store m
3  load m
4  store n
```

위의 경우는 비밀변수 x의 정보가 m으로 누출이 되고 다시 n으로 누출된다. 이러한 정보누출을 전이 누출(transitive leak)이라고 한다.

## 3. JVML0

본 논문의 분석 대상 언어는 JVML(Java Virtual Machine Language)의 부분언어인 JVML0[12]에 부울 연산자를 추가한 언어이다. JVML0의 명령어는 op, pop, push, load, store, if, goto, jsr, ret, halt로 구성되어 있고, 추가된 부울 연산자는 le, neg, and, true, false, eq 이다. 여기서 부울 연산자는 단순히 프로그램 작성의 편리함을 위해 추가한 것이므로 그대로 JVML0라고 부르기로 한다.

### 3.1. JVML0의 의미구조

JVML0의 명령어 프로그램 *code*는 op, pop, push, store, load, if, jsr, goto, ret, halt의 나열이며, *code*[*i*]는 *i* 번째 주소에 있는 명령어를 뜻한다 ($i \in Loc = \mathbb{N}$(자연수 집합)).

의미를 표현하기 위해서 프로그램의 실행상태(configuration) 집합 *Config*는 다음과 같다.

$$Config = Loc \times Mem \times Stack$$

여기서 *Mem*은 메모리의 집합을, *Stack*은 스택의 집합을 나타낸다. 즉,

$$Mem = Var \rightarrow \mathbb{Z}(\text{정수 집합})$$

$$Stack = \{\varepsilon\} \cup \mathbb{Z} \times Stack$$

프로그램의 의미는 다음과 같이 실행상태의 전이 →로 표현한다.

$$\rightarrow \subseteq Config \times Config$$

JVML0의 의미구조는 <그림 1>과 같이 정의한다. 여기에서 스택 $(k,s)$는 편의상 $k \cdot s$로 표기한다.

**[op]** $\langle i, m, k_1 \cdot k_2 \cdot s\rangle \rightarrow \langle i+1, m, (k_1 \text{ op } k_2) \cdot s\rangle$
where $code[i]$ = op

**[pop]** $\langle i, m, k \cdot s\rangle \rightarrow \langle i+1, m, s\rangle$
where $code[i]$ = pop

**[push]** $\langle i, m, s\rangle \rightarrow \langle i+1, m, k \cdot s\rangle$
where $code[i]$ = push $k$

**[load]** $\langle i, m, s\rangle \rightarrow \langle i+1, m, k \cdot s\rangle$
where $code[i]$ = load $x$ and $k = m(x)$

**[store]** $\langle i, m, k \cdot s\rangle \rightarrow \langle i+1, m[k/x], s\rangle$
where $code[i]$ = store $x$

**[if$_{false}$]** $\langle i, m, 0 \cdot s\rangle \rightarrow \langle i+1, m, s\rangle$
where $code[i]$ = if $j$

**[if$_{true}$]** $\langle i, m, k \cdot s\rangle \rightarrow \langle j, m, s\rangle$
where $code[i]$ = if $j$ and $k \neq 0$

**[goto]** $\langle i, m, s\rangle \rightarrow \langle j, m, s\rangle$
where $code[i]$ = goto $j$

**[jsr]** $\langle i, m, s\rangle \rightarrow \langle j, m, (i+1) \cdot s\rangle$
where $code[i]$ = jsr $j$

**[ret]** $\langle i, m, s\rangle \rightarrow \langle m(x), m, s\rangle$
where $code[i]$ = ret $x$

<그림 1 JVML0의 의미구조>

## 4. 안전성 분석

본 절에서는 주어진 바이트코드가 실행 중에 정보누출이 발생할 수 있는지 여부를 요약해석을 이용하여 분석하는 방법을 설명한다.

요약해석이란 프로그램과 같은 동적인 시스템의 실행 의미를 근사적으로 요약하는 이론으로서 주어진 시스템의 요약 의미구조로부터 프로그램 분석 알고리즘을 구현할 수 있다[5].

### 4.1 요약 프로그램

요약해석을 위해서 우선 구체 정의역(concrete domain)이 아닌 요약 정의역(abstract domain)의 값들로 바꾸는데, 본 논문에서 사용할 요약 정의역은 보안수준의 집합 $Sec$이다. 요약해석하는 동안에 변수들의 보안 수준은 $L : Var \rightarrow Sec$ 와 같은 요약 상태함수를 통해 유지된다. 이렇게 구한 요약 정의역을 가지고 요약연산을 통해 그 결과를 구할 수 있다.

JVML0 프로그램의 정보 흐름 안전성을 검사하기 위해 보안수준으로 표현하기 위해 요약 정의역에서 프로그램의 실행을 근사하는 요약해석을 수행을 하고자 한다. 이 JVML0는 public, secret 두 개의 보안수준을 가진다. 이때 상수는 public이고 각각의 변수는 각 보안수준의 값을 가진다.

### 4. 2 요약해석

JVML0을 가지고 요약해석을 함으로써 정보누출의 여부를 알 수가 있다. <그림 2>에서 보여주는 규칙들과 정의들은 C. Bernardeschi와 N. De Francesco가 제어 흐름 그래프의 정보를 이용하여 바이트코드를

**ipd**$^{\#}$ $\langle pc, m^{\#}, s^{\#}, \sigma, \rho\rangle \rightarrow \langle pc, m^{\#}, s^{\#}, \tau, \rho'\rangle$
where $\rho = (pc, \tau) \cdot \rho'$

**op**$^{\#}$ $\langle pc, m^{\#}, \tau_1 \cdot \tau_2 \cdot s^{\#}, \sigma, \rho\rangle \rightarrow \langle pc+1, m^{\#}, (\tau_1 \sqcup \tau_2) \cdot s^{\#}, \sigma, \rho\rangle$
where $c[i]$=op $i$ *not_in* $\rho$

**pop**$^{\#}$ $\langle pc, m^{\#}, \tau \cdot s^{\#}, \sigma, \rho\rangle \rightarrow \langle pc+1, m^{\#}, s^{\#}, \sigma, \rho\rangle$
where $c[i]$ = pop $i$ *not_in* $\rho$

**push**$^{\#}$ $\langle pc, m^{\#}, s^{\#}, \sigma, \rho\rangle \rightarrow \langle pc+1, m^{\#}, \sigma \cdot s^{\#}, \sigma, \rho\rangle$
where $c[i]$ = push $k$ $i$ *not_in* $\rho$

**load**$^{\#}$ $\langle pc, m^{\#}, s^{\#}, \sigma, \rho\rangle \rightarrow \langle pc+1, m^{\#}, (\tau \sqcup \sigma) \cdot s^{\#}, \sigma, \rho\rangle$
where $c[i]$ = load $x$ $m^{\#}(x) = \tau$ $i$ *not_in* $\rho$

**store**$^{\#}$ $\langle pc, m^{\#}, \tau \cdot s^{\#}, \sigma, \rho\rangle \rightarrow \langle pc+1, m^{\#}[\tau/x], s^{\#}, \sigma, \rho\rangle$
where $c[i]$ =store $x$ $i$ *not_in* $\rho$

**goto**$^{\#}$ $\langle pc, m^{\#}, s^{\#}, \sigma, \rho\rangle \rightarrow \langle j, m^{\#}, s^{\#}, \sigma, \rho\rangle$
where $c[i]$ = goto $j$ $i$ *not_in* $\rho$

**if**$^{\#}$ $\langle pc, m^{\#}, \tau \cdot s^{\#}, \sigma, \rho\rangle \rightarrow \langle j', m^{\#} \uparrow \tau, s^{\#} \uparrow \tau, \tau, (ipd(pc),\sigma)::\rho\rangle$
where $c[i]$ = if $j$, $(pc, j') \in E$(즉 이음선이 존재하면), $i$ *not_in* $\rho$

**jsr**$^{\#}$ $\langle pc, m^{\#}, s^{\#}, \sigma, \rho\rangle \rightarrow \langle pc, m^{\#}, ((pc+1),\sigma) \cdot s^{\#}, \sigma, \rho\rangle$
where $c[i]$ = jsr $j$ $i$ *not_in* $\rho$

**ret**$^{\#}$ $\langle pc, m^{\#}, s^{\#}, \sigma, \rho\rangle \rightarrow \langle pc, m^{\#} \uparrow (\sigma \sqcup \tau), s^{\#} \uparrow (\sigma \sqcup \tau), \sigma \sqcup \tau, (ipd(pc),\sigma)::\rho\rangle$
where $c[i]$ = ret $x$, $m^{\#}(x) = \tau$, ( $pc, j' \in E$ ) )

<그림 2 JVML0의 요약 의미구조>

분석하는 방법[10]에 기반하고 있다.

<그림 2>는 요약해석을 위한 변환규칙이다. 각 상태는 $\langle pc, m^{\#}, s^{\#}, \rho\rangle$로 이루어지고, 여기에서 각각의 요소들은

$pc$ : N (프로그램카운터)

$m^{\#} \in Var \rightarrow Sec$

$s^{\#} \in S^{\#} = \{\varepsilon\} \cup (Sec \times S^{\#})$

$\rho$ : $ipd$

을 나타낸다. 여기서 $ipd$는 제어 흐름 정보를 미리 계산한 값을 나타내는데 다음과 같이 정의된다. $n$개의 명령어로 이루어진 프로그램의 제어흐름 그래프상의 두 노드 $i$와 $j$가 있다고 가정하자(여기서 $j \neq i$). $j$로 들어가는 모든 경로가 $i$로부터 시작한다면, "node $j$ postdominates $i$"라고 하고 $j$ $pd$ $i$라고 쓴다. 그리고 $j$ $pd$ $i$이고 $j$ $pd$ $r$ $pd$ $i$ 같은 노드 $r$이 존재하지 않는다면 "$j$ immediately postdominates $i$"라고 하고 $j$ $ipd$ $i$라고 표기하는데 $j = ipd(i)$라고도 쓴다.

$W = \{x \mid c[j] = \texttt{store}\ x,\ j$는 $i$에서 시작하고 $ipd(i)$에서 끝나는 제어 흐름 그래프의 경로$\}$ 이고, $x \in W$에 대해서 $m^{\#}(x) = \sigma$라고 가정을 하자. 그러면 $m^{\#}(x)$에 $\tau$로 업그레이드하는 것을 $m^{\#}(x) \uparrow \tau$라고 한다면 $\sigma \sqcup \tau$을 가지게 된다.

또한 $s^{\#} \in S^{\#}$ 이라고 가정하면 $s^{\#} \uparrow \tau$는 스택 $s^{\#}$에 있는 값에 $\tau$를 업그레이드하는 것을 나타내고 다음과 같이 정의된다.

$s^{\#} \uparrow \tau$ = if $s^{\#} = \varepsilon$ then $\{\varepsilon\}$

otherwise $(h \sqcup \tau) \cdot (s' \uparrow \tau)$

(i.e., $s^{\#} = h \cdot s'$)

묵시적인 정보흐름이 발생하면 그 흐름이 끝날 때까지 스택에 들어있는 값들은 이미 묵시적인 정보흐름이 발생한 시점에서의 값에 영향을 받는다. 왜냐하면 제어 흐름의 경로가 어느 쪽으로 결정이 되든 묵시적인 정보흐름이 발생하는 시점에서의 값에 따라 스택의 값이 결정되기 때문이다.

## 5. 실험

이 장에서는 실험대상으로 삼은 While 프로그램을 가지고 JVML0로 바꾸고 그것을 요약해석을 통해 정보누출의 여부를 검사하는 분석기의 구현에 대하여 설명한다.

### 5.1. Whlie 언어

1) While의 구문 구조

본 논문에서 정보흐름의 안전성을 검증하기 위한 대상이 되는 프로그램은 기본적인 명령형 언어인 While 의 프로그램이며 그 문법의 구조는 배정문, 조건문, while문 등을 정의한 것으로 다음과 같다.

$S$ ::= $x := e$ | if $e$ then $S$ else $S$ | $S$ ; $S$
| while $e$ do $S$ | skip

$e$ ::= $c$ | $x$ | $e$ $op$ $e$

2) While 프로그램을 JVML0로 변환

실험을 하기 위한 While 프로그램을 검사하기 위해 JVML0로 변환을 하는데 그 변환 규칙은 <그림 3>과 같다.

$CS$ : Stmt → N → $Code$× N
$CS$ [ $x$ := $a$ ] $pc$ = $CA$ [ $a$ ] $pc$ ⊕ (store $x$, 1)
$CS$ [ skip ] $pc$ = (ε, 0)
$CS$ [ $S_1$ ; $S_2$ ] $pc$ = $CS$ [ $S_1$ ] $pc$ ⊕ $CS$ [ $S_2$ ] $pc'$
where $CS$ [ $S_1$ ] $pc$ = ($c$, $l$) ⇒ $pc'$ = $pc$ + $l$
$CS$ [ if $b$ then $S_1$ else $S_2$ ] $pc$ = $CB$ [ ¬$b$ ] $pc$
: if $j_1$ : $CS$ [ $S_1$ ] $pc_1$ : goto $j_2$ : $CS$ [ $S_2$ ] $pc_2$
where $CS$ [ ¬$b$ ] $pc$ = ($c$, $l$), $CS$ [ $S_1$ ] $pc_1$ = ($c_1$, $l_1$),
$CS$ [ $S_2$ ] $pc_2$ = ($c_2$, $l_2$), $j_1$ = $pc$ + $l$ + $l_1$ + 2,
$pc_1$ = $pc$ + $l$ + $l1$, $j_2$ = $pc_1$ + $l_1$ + $l_2$ + 1
$CS$ [ while $b$ do $S$ ] $pc$
= $CB$ [ ¬$b$ ] $pc$ : if $j$ : $CS$ [ $S$ ] $pc_1$ : goto $pc$
where $CS$ [ ¬$b$ ] $pc$ = ($c$, $l$), $CS$ [ $S$ ] $pc_1$ = ($c_1$, $l_1$)
$j$ = $pc$ + $l$ + $l_1$ + 2, $pc_1$ = $pc$ + $l$ + 1

<그림 3 While-to-JVML0 번역기>

<그림 3>의 변환식에 나타난 $CS$에서 ($c_1$, $l_1$) ⊕ ($c_2$, $l_2$) = (($c_1$ : $c_2$), ($l_1$ + $l_2$)) 이고, 이때 "⊕" 는 컴파일된 두 결과의 합성, ":" 는 두 개의 순차적인 명령어사이의 연결을 나타낸다.

3) 표현식의 변환

위의 변환규칙에서 $CS$에는 산술연산인 $CA$와 부울연산인 $CB$가 있는데 그 각각의 변환규칙은 <그림 4>와 같이 정의한다.

$a$ ∈ $Aexp$ 산술식
$b$ ∈ $Bexp$ 부울식
$CA$ [ $n$ ] = (push $n$, 1)
$CA$ [ $x$ ] = (load $x$, 1)
$CA$ [ $a_1$ + $a_2$ ] = $CA$ [ $a_2$ ] ⊕CA [ $a_1$ ] ⊕ (add, 1)
$CA$ [ $a_1$ − $a_2$ ] = $CA$ [ $a_2$ ] ⊕CA [ $a_1$ ] ⊕ (sub, 1)
$CA$ [ $a_1$ * $a_2$ ] = $CA$ [ $a_2$ ] ⊕CA [ $a_1$ ] ⊕ (mul, 1)
$CB$ [ $a_1$ ≤ $a_2$ ] = CA [ $a_2$ ] ⊕CA [ $a_1$ ] ⊕ (le, 1)
$CB$ [ ¬$b$ ] = $CB$ [ $b$ ] ⊕ (neg, 1)
$CB$ [ $a_1$ ∧ $a_2$ ] = $CA$ [ $a_2$ ] ⊕$CA$ [ $a_1$ ] ⊕ (and, 1)
$CB$ [ $true$ ] = (1, 1)
$CB$ [ $false$ ] = (0, 1)
$CB$ [ $a_1$ = $a_2$ ] = $CA$ [ $a_2$ ] ⊕$CA$ [ $a_1$ ] ⊕ (eq, 1)

<그림 4 표현식들의 변환규칙>

## 5.2. 실험 예

아래에는 While 프로그램을 가지고 실험을 한 결과의 한 예가 나와 있다.

if $y$=o then $x$:=1 else $x$:=0

| | | |
|---|---|---|
| 1 | load y | $ipd$(1) = 2 |
| 2 | if 5 | $ipd$(2) = 6 |
| 3 | push 1 | $ipd$(3) = 4 |
| 4 | goto 6 | $ipd$(4) = 5 |
| 5 | push 0 | $ipd$(5) = 6 |
| 6 | store x | $ipd$(6) = 7 |
| 7 | halt | $ipd$(7) = 8 |
| 8 | end | |

(a) JVML 0 (b) $ipd$

여기서

$m^{\#}(x)$ = public
$m^{\#}(y)$ = secret

라고 가정하면 분석결과는 다음과 같다.

<1, $m^{\#}$, Ɛ, public, Ɛ>
<2, $m^{\#}$, secret • Ɛ, public, Ɛ>
<3, $m^{\#}$[secret/$x$], Ɛ, secret, (6,public) • Ɛ>
<4, $m^{\#}$[secret/$x$], secret • Ɛ, secret, (6,public) • Ɛ>
<6, $m^{\#}$[secret/$x$], secret • Ɛ, secret, (6,public) • Ɛ>
<6, $m^{\#}$[secret/$x$], secret • Ɛ, public, Ɛ>
<7, $m^{\#}$[secret/$x$], Ɛ, public, Ɛ>

(1) $y$=0 의 implicit flow

<1, $m^{\#}$, Ɛ, public, Ɛ>
<2, $m^{\#}$, secret • Ɛ, public, Ɛ>
<5, $m^{\#}$[secret/$x$], Ɛ, secret, (6,public) • Ɛ>
<6, $m^{\#}$[secret/$x$], secret • Ɛ, secret, (6,public) • Ɛ>
<6, $m^{\#}$[secret/$x$], secret • Ɛ, public, Ɛ>
<7, $m^{\#}$[secret/$x$], Ɛ, public, Ɛ>

(2) $y$≠0 의 묵시적 흐름

(1), (2) 의 경우 x의 보안수준이 public에서 y의 보안수준인 secret으로 변화된 것을 알 수 있다. 그러므로, 위의 주어진 프로그램의 예는 정보누출이 있는 프로그램이라고 할 수 있다.

## 6. 결 론

본 논문은 명령형 언어 While 프로그램에 대하여 정보흐름 안전성을 검증하기 위하여 이를 바이트코드로 바꾸고 이것을 가지고 요약 해석을 이용하는 방법을 설명하였다. 본 논문의 실험을 통해 원시 프로그램을 대상으로 하는 분석 방법[13]이 바이트코드를 대상으로 할 때에도 제어흐름도의 정보만 있으면 그대로 적용할 수 있다는 것을 알 수 있다.

실용적인 분석기를 만들기 위해서는 본 논문에서는 다루어진 JVML0 코드를 JVML로 확장하여 적용하는 것이 필요하다. 이를 위해서는 프로시저간의 분석법을 연구해야 할 것이다.

## 참고문헌


[1] D.E. Denning, “A Lattice Model of Secure Information Flow”, Communications of the ACM, 19(5):236-242, 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.A.Goguen and J.Meseguer, Unwinding and inference control, In Proc. IEEE Symp. on Security and Privacy, pp. 75-86, 1984.

[4] 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, 1984.

[5] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, p.p 238-252, 1977

[6] 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.

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

[8] R. Joshi and K.R.M. Leino, A semantic approach to secure information flow, Science of Computer Programming, 37, pp.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] C. Bernardeschi, and N. De Francesco, "Checking Security of Java Bytecode by Abstract Interpretation", Proceedings of the 17th symposium on Proceedings of the 2002 ACM symposium on applied computing , 229-236, 2002

[11] 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.

[12] R. Stata and M. Abadi, "A Type System for Java Bytecode Subroutines", ACM Trans. Program. Lang. Syst., 21(1):90-137, 1999

[13] 신승철, 도경구, 이수호, "요약해석을 이용한 정보흐름 제어", 정보과학회 학술발표대회, 2002, pp.355-357

## 조 순 희

2000년 2월 동양대학교 컴퓨터공학과(학사)
2001 년 3월 ~ 현재 동양대학교 컴퓨터공학과 석사과정재학
관심분야는 프로그래밍 언어, 프로그램 분석 및 검증

## 신 승 철

1987년 인하대학교 전자계산학과(학사)
1989년 인하대학교 전자계산학과(석사)
1996년 인하대학교 전자계산공학과(박사)
1999년 9월 ~ 2000년 9월 Postdoc Researcher, Kansas State University.
1996년 ~ 현재 동양대학교 컴퓨터공학부 조교수.
관심분야는 프로그래밍 언어, 프로그램 분석 및 검증, 정형기법, 이론 전산학.

## 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 프로그래밍언어, 프로그램 분석 및 검증, 의미론.