자바 바이트 코드에서 정보보호 보안성 검사
박성훈, 이정림, 권기현
경기대학교 정보과학부
{sachem0, leejrim, kkhwon}@kyonggi.ac.kr

Checking Security of Information Flow on Java Byte Code
Sachoun Park, Jungrim Lee, Gihwon Kwon
Department of Computer Science, Kyonggi University

요약
현재 정보보호시스템의 여러 저점은 보안성에 대한 만연한 문제가 되고 있는데 이러한 시스템의 이해와 보안성이 특히 우려되고 있다. 특히 정보보호시스템의 보안성에 대한 연구는 다양한 이유로 보안성에 대한 중요성이 인식되고 있다. 이는 보안성 shortage에 대한 이해와 보안성에 대한 중요성의 인식이 요구되는 시점이다. 따라서 본 연구는 정보보호시스템의 보안성에 대한 검사를 위한 방법론을 제시한다.

1. 서 론
본 연구의 목적은 정보보호시스템의 사용자로부터 여러 정보를 제공하는 시스템에서의 정보 보안을 구현하는 것입니다. 특히 정보보호시스템의 보안성에 대한 연구는 다양한 시점에서의 중요성이 인식되고 있다. 본 연구는 이러한 시점에서의 정보보호시스템의 보안성에 대한 연구를 위한 방법론을 제시한다.

또한 본 연구의 비 보안성 측면에 대한 연구로, 프로그램의 보안성에 대한 연구는 보안성에 대한 만연한 문제로 인해 사용자로부터 보안성에 대한 중요성이 인식되고 있다. 본 연구는 이러한 시점에서의 보안성에 대한 연구를 위한 방법론을 제시한다.

또한 본 연구의 비 보안성 측면에 대한 연구로, 프로그램의 보안성에 대한 연구는 보안성에 대한 만연한 문제로 인해 사용자로부터 보안성에 대한 중요성이 인식되고 있다. 본 연구는 이러한 시점에서의 보안성에 대한 연구를 위한 방법론을 제시한다.

2. 관련 연구

* 본 연구는 2004년 한국정보보호학회 전자정보보호학회 보안성 모델 평가방법론 연구 지원에 의하여 수행되었음.

3. 바이트코드의 구문과 의미
3.1 프로그램 구문
본 논문에서 사용할 바이트코드와 코드에 대한 간단한 설명은 아래와 같다.

op: 스택에서 두개의 오퍼랜드를 빼내서 (POP) 연산을 수행하고 그 결과를 다시 스택에 넣는다 (push). 여기서 연산자인 산발산 연산 면명어들 (add, sub, div, mul)과 눈리산산 연산 면명어들 (and, or, xor)을 대표한다.

pop: 스택의 맨처음(top)의 값을 가져온다.

push k: 숫자 k의 값을 스택에 씁니다. 이 명령어는 수치를 스택에 씁는 다른 명령어들 (bipush, sipush, iconst_m1, iconst_0, iconst_1, iconst_2, iconst_3, iconst_4, iconst_5, ldc, ldc_w)를 대표한다.

load i: 지역변수 배열 m의 번째 값 m(i)를 스택에 씁니다. 이 명령어는 변수를 스택에 씁는 모든 명령어들 (aload, aaloed, aaloed_1, aaloed_2, aaloed_3, iload, iload_0, iload_1, iload_2, iload_3)을 대표한다.

store i: 스택의 최단간의 값을 빼서 지역변수 배열 m(i)에 그 값을 씁니다. 이 명령어는 다음 명령어들로 대표된다: astore, astore_0, astore_1, astore_2, astore_3, astore_4, astore_5, istore, istore_0, istore_1, istore_2, istore_3.

if: 스택의 최단간의 값을 0과 비교해서 1의 위치로 프로그램 캐언터를 옮기는 명령어들 (ifeq, ifne, ifge, iflt, ifull, ifnonnull)를 대표한다.

if cmp: 스택의 두 값을 비교해서 0의 위치로 프로그램 캐언터를 옮기는 명령어들 (if_icmpeq, if_icmpne, if_icmpgt, if_icmplt, if_acmpg, if_acmple, if_acmpgt, if_acmple, if_acmpgt, if_acmple)를 대표한다.

goto: 단순히 프로그램 캐언터의 위치로 옮긴다.

nop: 아무것도 수행하지 않는 명령이다. 실제로 이 명령은 neg, rem, shl, shr, ushr, 2b, 2s, nop 등을 대표한다. 왜냐하면 단순 연산자들은 정보보안의 보안성에 아무런 영향을 주지 않는다고 판단하기 때문이다.

return: 이 명령어는 실제 메소드의 수행이 종료하면 메소드를 호출했던 스택 프로그램을 끝낸다는 명령어이다. 하지만 우리는 단순히 메소드의 종료를 알리는 명령으로 사용하였다. 이 명령어 역시 return, ireturn, areturn, areturn의 명령들로 대표된다.

여기서 다루는 바이트코드 명령어는 기본적으로 자바관
드 가상머신의 명령어 집합을 기반한다 [7]. 제외된 명령
들은 예외상황을 다루는 jsr/fret 명령과 랜덤지연 명령, 범위
관련 명령어들. 자바관 가상머신의 모든 명령어를 다루는 것
은 향후연구가 될 것이 있다.

3.2 프로그램의 의미
우리는 프로그램의 의미를 전자 시스템으로 표현한다. 그 결과는 프로그램의 실행 결과이며 각 노드는 프로그램의 상호적 즉 스택과 지역변수 배열의 변화를 나타낸다. 이 프로그램의 기반으로는 스택의 인터레이크나 오버리크로가 존재하지 않는다면 goto 명령어나 if 명령어에 의해서 존재하지 않는 프로그램 캐언터로 전환할 수 없다는 것이다.

정의 1. 전자 시스템은 $T = (s, R, S)$이다. 여기서 $S$는 상태들의 집합이고, $s_0 \in S$는 초기 상태를 나타낸다. $R \subseteq S \times S$는 전이관계

 bev를 위한 예제를 고려하고자 한다. bev의 경우 레벨이 그림 1과 같다. 그림에서 bev의 값에 대한 도메인 $C$는 데이터 ($k, k(i)$)와 프로그램 캐언터 ($f, k, b$)를 위해 사용되었다. 그 프로그램에서 사용되는 변수의 접근법은 bev의 변수를 주는 함수 $M \ast V = C'$에 $m, m', m''$ 등이 있
다. 그리고 여기에 대한 유한 상태변수의 스택에 대한 도메인

$S = (C')^* = s, s', s''...$ 등이 있다. 주변적 지역변수 배열 bev에 대해서는 $D(m) \subseteq C'$로 정의한다.

실행되는 프로그램의 상태는 트플 $< i, m, p >$로 표현한다. 여기서 $i$는 프로그램 캐언터를 의미하고, $m$은 지역변수의 현재 값이다. $s$는 오퍼랜드 스택의 현재 상태를 가리킨다. bev를 변형 명령어라

고 한다면 항상 프로그램의 시작은 bev[0]이고, 스택은 비어있으므로 가정한다. 지역변수 배열 bev에는 첫 번째 변수로부터 빈번 변수가 차례로 저장되고 나머지 변수는 정리되지 않은 것으로 가정한다. $O$를 상태들의 도메인이라고 한다면, 프로그램의 의미의 구조들은 진입관계 $\leq C$를 정의한다. 프로그램 bev[0]는 $m$의 인덱스

에 해당하는 값을 $s'$로 변경하는 것을 의미한다. $cmp(k, k)$는 두 값을 비교하는 스크립트이다. 주변 bev된 바이트코드의 프로그램 bev와 지역변수 배열 bev에 대해서, 프로그램의 현재 호흡은 진입 시점
(O\frown i, m, λ)으로 정의된다. 여기서 $i$는 번째로부터 시작한다. $cmp$의 명령어를 의미한다. 시점의 종합 (concatenation) 연산자는 $s'$를 이용하여 생성하기 위해 사용되었다. 프로그램의 마지막 상태는 유일하게 사용하기에 자기 자신에게 전이가 발생한다. 그 명령어는 bev[0] = return으로 된다.

<table>
<thead>
<tr>
<th>명령</th>
<th>명령어</th>
<th>명령어</th>
<th>명령어</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>bydata[i] = op</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pop</td>
<td>bydata[i] = pop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>push</td>
<td>bydata[i] = push k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>load</td>
<td>bydata[i] = load i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>store</td>
<td>bydata[i] = store i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if true</td>
<td>bydata[i] = if $\text{if cmp}[k, k]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if false</td>
<td>bydata[i] = if $\text{if cmp}[k, k]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>goto</td>
<td>bydata[i] = goto</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td>bydata[i] = nop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>return</td>
<td>bydata[i] = return</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

그림 1. 프로그램의 의미

4. 바이트코드 프로그램 모델

보안 속성을 기술하기 위해서 우리는 바이트코드 프로그램의 각

영역에 대한 보안 레벨을 부여한다. 보안 레벨은 $low$, $high$을 두개의 범위로 정의한다. 보안 레벨 $low$, $high$은 두개의

원소로 가지는 집합이다. 보안 레벨을 부여하는 함수 $level$은 프로

1. 바이트코드 프로그램 모델

보안 레벨을 부여하기 위해서 우리는 바이트코드 프로그램의 각

영역에 대한 보안 레벨을 부여한다. 보안 레벨은 $low$, $high$을 두개의

원소로 가지는 집합이다. 보안 레벨을 부여하는 함수 $level$은 프로

1. 바이트코드 프로그램 모델

보안 레벨을 부여하기 위해서 우리는 바이트코드 프로그램의 각

영역에 대한 보안 레벨을 부여한다. 보안 레벨은 $low$, $high$을 두개의

원소로 가지는 집합이다. 보안 레벨을 부여하는 함수 $level$은 프로

1. 바이트코드 프로그램 모델

보안 레벨을 부여하기 위해서 우리는 바이트코드 프로그램의 각

영역에 대한 보안 레벨을 부여한다. 보안 레벨은 $low$, $high$을 두개의

원소로 가지는 집합이다. 보안 레벨을 부여하는 함수 $level$은 프로
그림의 의미에 의해서 다음과 같이 정의된다.

\[
\begin{align*}
\text{level}(k) &= \begin{cases} 
0, & \text{low} \\
2, & \text{otherwise}
\end{cases} \\
\text{level}(m(k)) &= \begin{cases} 
0, & \text{low} \\
1, & \text{otherwise}
\end{cases} \\
\text{level}(m(k,l)) &= \begin{cases} 
0, & \text{low} \\
1, & \text{otherwise}
\end{cases}
\end{align*}
\]

여기서 \(m\)은 최상 하산 (least upper bound)으로의 의미를 갖는다. 이때 바이트코드의 프로그램에 대한 모델을 크리키 구조 K로 정의할 수 있다.

정제 2. 크리키 구조 \(K = (Q, q_0, R, Label, AP)\)이다. \(R\)는 상태들 간의 직접으로 프로그램의 구문과 프로그램 캐러터에 대해서 \(q_0\)래와 같이 정의한 \(PC,INS,LEVEL\)로 정의된다. \(K\)과 \(K\)는 각각 스택에 최상단의 값과 그 아래의 값을 나타낼 수 있다.

\[
\begin{align*}
q(<>p) &= (q, p, level(k, opk)) \\
q(><p) &= (q, p, \text{pop}) \\
q(><push) &= (q, p, \text{push}, \text{low}) \\
q(><laad) &= (q, i, level(m)) \\
q(><store) &= (q, i, level(m)) \\
q(i, j) &= (q, i, level(k)) \\
q(i, j, opk) &= (q, i, j, level(k)) \\
q(i, j, <) &= (q, i, j, \text{load}, opk) \\
q(i, j, >) &= (q, i, j, \text{store}, opk) \\
q(i, j, return) &= (q, i, j, \text{return})
\end{align*}
\]

\(\omega\)는 초기상태이고, \(\text{level}(q)\)는 정제관계이며 전체 관계 (total relation)이다. \(\text{level}(q)\)는 \(q_1\)로도 정의한고 \(q_2\)는 \(q\)의 바로 다음에 수정되는 명령이다. 이전 정제는 바이트코드 명령에 따라 결정적으로 종결된다. \(Label, Q \rightarrow 2Q\)는 상태에 환산 명령의 부분집합을 레이블 하는 함수이다.

5. 정화효과의 보안 속성 검사

정화효과의 보안은 정화 효율이 보안성과 프로그램 종류에 관계된 보안성으로 나누어 볼 수 있다. 정화효과에 대한 보안성은 다시 명령형의 효율과 명시적인 효율로 나란다. 고차원의 언어에서 명령형의 효율에 대한 조사는 배경이 발생하는 곳을 조사하여 배경을 오른쪽의 보안 레벨이 어려울지 않아야 하는 것으로 해석되었다. 그러나 파이언코드에서 파이언코드를 사용하기 때문에 우리는 지역 변수에 저장하는 store 명령을 중심으로 명시적 정화효과에 관해서 조사한다. 명시적 정화효과는 확장 복잡한 측면이 있다. 고차와의 언어에서 구조적 자료가 유사하거나 따른 \(\text{level}(q)\)의 영역을 쉽게 확인할 수 있고 그 영역 안에서 검사가 완료될 수 있다. 그러나 바이트코드로의 명령형의 영역을 간단하게 복잡하게 막으려고 한다. 왜냐하면 \(q_0\)에서 명령형의 프로그램의 캐러터와 \(q\)이 후에 다른 \(q\)에 있는 \(q_0\)의 명령형과 관계되어 있어서 정적 영역이 확장되기 때문이다. 마지막으로 프로그램 정의에 의한 정화노수 (covert flow)는 고차원 언어에서 while문에 나온 보안 레벨의 변수가 사용되는 것을 명시하지 않고 원래 

6. 결론 및 향후 연구

유리는 보안성에 극히 요구되는 분야인 삼차드코드에서 애플리케이션에서 정화효과의 보안성 문제를 다룬다. 본 연구는 정보보안 진화목의 과학수행의 일환으로 이루어졌으며, 공통평가는 이론과 기반의 높은 평가를 받기 위한 자바 버전 코드 진화이다. 실제 애플리케이션은 바이트코드를 제공하기 위해 앞으로 바이트코드로 변환될 때 애플리케이션에 대한 바이트코드 진화는 시점과 아우러 요구되는 연구 분야이다.

우리는 우선 진화형 프로그램에서 특정 배경 프로그램의 범위를 통해서 구문과 의미를 정의하고, 안전한 정보보안을 정의한 후에 프로그램에 대해 SMV 모델을 만들어 몇몇 예제에 대한 실행을 수행하였다.

본 연구는 현재 진행 중이며, 본 연구의 정화성의 중요성을 모델 자체 코드에서 사용하는 모든 바이트코드로 구문과 의미를 확장하고, 진화의 단계를 자동화하는 연구가 남았다.

참고문헌