소형 기기용 효율적인 바이트 코드 검증기술에 관한 연구

황 철 준, 조 중 보, 정 민 수
경남대학교 컴퓨터공학과

An Efficient Bytecodes Verification Technology for Small Devices

Cheul-Jun Hwang, Jeung-Bo Cho, Min-Soo Jung
Dept of Computer Engineering, Kyungnam University

요약
USIM(Universal Subscriber Identity Module) 기술은 데이터의 관리와 보호를 하나의 소형 칩 내에 집약시킨 기술로 최근에 이 기술에 대한 관심도가 증가하고 있다. 자바카드 가상기계는 자바 언어로 작성된 프로그램이 USIM 칩 내부에서 실행 가능하도록 해준다. 하지만 자바카드 가상 기계는 자원 제약적인 디바이스의 특성으로 인하여 검증기술을 제공하지 않는다. 본 논문에서는 정적 체크 기반의 효율적인 바이트코드 검증기를 제안한다. 연구 결과 본 논문에서 제안하는 검증기는 일반적인 것에 비해 90% 이상의 사이즈가 감소되었다.

1. 서론

자바카드언어는 다양한 스마트 카드 어플리케이션의 프로그래밍을 위한 보편적인 프로그래밍언어이다. 자바 카드 포럼에 의하면, 자바카드 분야에서의 USIM 기술은핵심적인 역할을 한다. 자바카드 언어는 아래의 두가지의 중요한 특성 때문에 스마트카드를 위한 이상적인 플랫폼임을 알 수 있다. 첫번째로 자바카드 언어는 자바 언어의 하부구조로 이루어져 있으며 자바카드 어플리케이션 프로그래밍 인터페이스 (APIs)를 사용한다는 점이다. 자바카드 개발자들은 자바 기술에서 제공하는 장점들을 쉽게 획득하여 프로그래밍에 사용할 수가 있다. 두번째로 자바카드 보안 모델은 하나의 카드 상에 여러 개의 어플리케이션들이 적재되는 것과 이들 상호간에 안전한 통신을 지원한다. 이러한 원리에 의해서 카드 발급 후에도 애플릿이라고 불리는 새로운 어플리케이션 프로그램의 적재도 가능해진다. 스마트카드는 어플리케이션의 확장이 가능한데 이러한 사후 발급에서 가장 중요한 부분은 보안에 관한것이다. 만약에 사용자가 원치않는 불필요한 애플릿이 로딩 되어진다면 그 순간부터 이 애플릿으로 인하여 자바카드 가상머신의 보안성은 크게 저하될것이다. 간단한 예를 들어 설명하면 불필요한 애플릿은 스택의 오브플로를 일으켜 메모리의 위치를 임의로 변경할 수 있다. 뿐만 아니라 승인되지 않은 접근이 이루어질 경우에 그것은 불필요한 객체를 생성하며 메모리의 변경이나 에러를 발생하거나 심지어 다른 프로그램까지도 변경할 수도 있다. 일반적인 보안 관련 문제들은 이러한 사후에 발급되어진 애플릿에 의해 발생되어지는 것은 당연한 일이다. 스마트 카드와 같은 자원 제약적인 장치에서 자바의 보안 모델을 적용하는 것은 다른 장치와 달리 보다 세밀하고 정교한 보안시스템을 필요로 한다.[1][2][9]

2. 관련연구

2.1 자바 카드

자바카드는 자바언어의 하위구조이면서 정형적인 검증을 수행하는 이상적인 언어이며 산업 어플리케이션에 적용 가능한 적절한 크기의 언어라고 볼수가 있다. 자바와 비교해볼 때 자바카드 가상기계는 멀티스레딩과 동적인 클래스로드등의 여러가지 기능에 대한 제약 사항이 존재한다. 자바와는 달리 자바카드에서의 실행파일은 클래스파일이 아니고 Converted APplet (CAP)이라고

불리는 변환되어진 애플릿이 실행 파일이 된다. 자바카드 컨버터의 수행 결과에 의해서 자바카드 프로그램은 캡파일로 변환되어진다. 자바카드 가상기계는 두개의 파트로 나뉘어져 있으며 이를 off-card on-card라고 한다.[1][3]

자바카드 컨버트는 그림.1에서 보듯이 off-card 부분에 속한다.

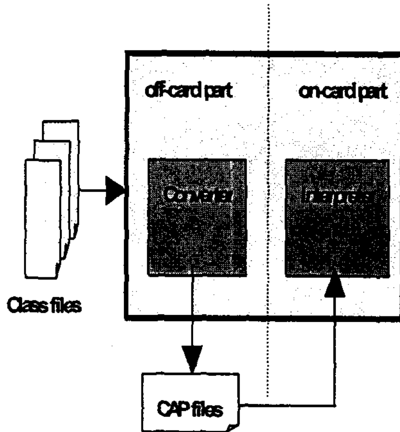

그림.1 자바카드 가상 머신의 구조

2.2 전통적 검증기

바이트코드 전체를 완전히 검증하는 작업은 많은 양의 리소스를 필요로 하는 작업이다. 스마트카드와 같이 매우 제약적인 디바이스 상에서 수행되는 것을 지원하기 위하여 제안되었다. 이러한 시스템은 외부에 위치하는 선 처리기에 의해서 이루어진다. 타입 검증은 (바이트 코드 검증과 유사함) 리소스를 가장 많이 소비하게 되는 검증 파트의 한 부분이므로, 검증 알고리즘을 단순화하기 위한 작업이 필요하다. 보통 2가지의 접근 방법이 일반적으로 사용되어지는데 이는 Bytecode normalization과 proof carrying code (PCC) 또는 이와 유사한 기술이다.[6][10]

2.3 바이트코드 정규화에 의한 검증

바이트코드정규화는 Trusted-Logic's의 스마트카드 검증기의 사용에 의해서 접근되어지는 방법이다. 이것은 검증되어진 애플릿을 정규화 하는것으로 구성 되어지므로 검증 절차가 보다 간단해진다. 보다 상세하게 말하자면, 애플릿이 생성되어질 때 그것은 각각의 고유한 객체이며 그 자신만의 타입을 가지고 있으며 스택은 비어있다. 이것은 메모리 요구를 크게 줄일 수 있는 것이며, 검증기는 각각의 명령어에 대해서는 타입 정보를 유지하지 않아도 된다. 하지만 검증되어진 메소드의 변수들은 단일해야만 한다. 리소스에대한 요구는 간소화 되어진 fixpoint 작업이 수행되어지고난 이후에야 감소 되어지는 것이다. 반면에 코드가 생성되었을 때는 그것의 크기와 메모리 요구량은 이론적으로는 증가할 것이다.

2.4 경량 바이트코드 검증

Necula 와 Lee에 의해서 소개되어졌고, proof-carrying이라는 코드는 프로그램에서 안전성을 높이기 위해서 추가 되어지는 코드로 구성되어진다. Proof는 코드 생성기 에의해서 생성되어지며, 그것의 안전한 Proof를 따라서 그 코드는 전송되어진다. 코드 수신기는 프로그램 안전을 확보하기 위하여 Proof를 검증할 수 있다. Proof를 체크하는 작업은 그것을 생성해 내는 작업보다 훨씬 간단하다. 검증 처리과정은 제약적인 요소를 가지는 장치(스마트 카드)에서 수행되어질수 있다. 이 기법은 현재 썬사의 KVM에서 사용되어지고 있다. Proof는 분기 타깃들을 위해 지역 변수들의 내용과 스택 요소에 부합한 추가적인 형 정보에 존재한다.

바이트코드 정규화에 비교하여 경량 검증은 jsr과 ret 명령어를 바이트코드와 형 정보의 저장을 위해서 EEPROM 메모리 내에 존재 하는 임시적인 저장 공간으로 부터 제거 하는 작업을 별도로 필요한다. 경량 검증은 코드를 통해서 linear pass와 유사하게 검증을 수행한다.[10]

3. 정적체크 기반의 검증기 설계의 주요 문제점

본 논문에서 제안한 검증기는 2스텝으로 나뉘어진다. 첫번째 스텝은 off-card에 존재하며 두번째 스텝은 on-card에 존재한다.

3.1 Step 1: off-card 상에서의 Proof Component 생성

정적 검사에 기반을 둔 검증기는 스택 기반의

ByteCodes가 의미론적으로 정확하는 지를 검증한다. 스택 명령어는 많은 자원을 사용하게 되는데 이는 그 자체의 복잡성에 기인한다. 이과정에서는 off-card 내에서의 스택상태의 Proof Component에 관해서 나타낸다. 아래에서 굵은 글씨체로 되어있는 6개의 바이트 코드들은 on-card에서 검증되어진다. 굵은 글씨체의 바이트 코드들이 실행되어지고 난 이후의 스택의 상태는 Proof Component에 저장되어진다. 검증기는 우선적으로 위에서 정한 6개의 바이트코드가 스택 상태에서 정확하게 사용되어졌는지를 체크한다. 바이트코드는 실제 값을 검증하는 것이 아니라 그것의 타입을 검증한다.

3.2 Step 2: on-card 부분에서의 검증

만약에 스택의 상태의 상태에대한 검사를 마친 컴포넌트가 on-card 부분에 전송되면, 6-ByteCodes는 검증기에에 의해서 검증 되어 질것이다.

on-card 부분에서의 검증은 다음의 Fig.5에서 볼수가 있다.(레지스트는 메소드의 지역변수를 저장하기 위해서 사용되어진다.)

3.3 검증기의 주요 알고리즘

자바카드에는 185바이트의 바이트코드가 존재한다. 자바카드 애플릿 내에서 사용되어지는 모든 바이트코드들은 카운트 되어진다. 사용 빈도가 높은 몇몇의 바이트 코드들은 표 1에서 알수 있다.

표 1.자바카드 바이트 코드 사용 빈도

ByteCodes	Counts	Ratio (%)
Store	43505	8.24
Return	36550	6.92
Const	35896	6.80
Push	27781	5.26
Load	136383	25.84
And	23618	4.47
Total	303733/527900	57.54

위의 6개의 바이트코드들은 50% 이상의 사용빈도를 보이므로 이것들을 검증하는 작업이 효율적임을 알수있다. 본 논문 에서의 검증기는 모든 바이트 코드를 체크하지 않고 위의 6개의 바이트 코드를 중심으로 검증 작업을 수행하게된다. 여기서 사용 되어지는 새로운 알고리즘은 Proof Carrying Code (PCC)과 유사하다. 자바카드 가상기계는 2개의 파트로 구성되어지며 off-card(자원 비제약적인 부분)와 on-card(자원 제약적인 부분)으로 구성되어진다. 스택에 관련된 정보들은 Proof Component라고 부르며, 이는 off-card 부분에서 생성되어지고 on-card 부분으로 전송되어진다. 이 정보들은 정적 체크 방식에 기초를 두고 있으며 바이트 코드 검증에서 사용되어진다. 이 방법은 검증기의 사이즈를 감소 시키도록 한다. 본 논문에서 제안하는 검증기는 그림.2에서 알 수 있다.[4][7]

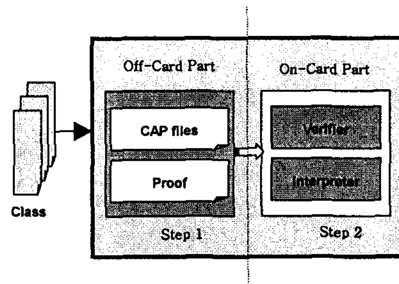

그림2. 검증기 구조

4. 검증기의 성능 비교

4.1 검증기간의 성능 비교

본 논문에서의 검증기는 2개의 부분으로 구성되어있다 (i) 스택의 상태들과 Proof Component를 생성하는 부분 (ii) 전송되어진 Proof Component를 이용하여 6바이트 코드들을 검증하는 단계이고 여기서 검증기는 off-card 영역을 말하는것이며 이것은 Proof Component와 CAP파일을 생성하는 것을 의미한다. CAP파일은 원본의 CAP파일보다 그 사이즈가 증가한다. 실질적으로 자바카드 애플릿의 총 크기는 대략 28.8KB 정도이다. Proof Component의 크기는 1.7%와 5.9%의 사이다. Proof Component의 평균 사이즈는 Table 2에서 보는것과 같이 4.35%이다.

표 3은 검증기들간의 성능을 비교해 놓은 것을 나타낸 것이다.[7][8][10]

표 2. off-card 부분에서 생성되는 Proof Component의 크기 비교

File Name (Class)	Size of Originals	Size of Proof Components	Ratio
A	7181 bytes	210 bytes	2.92 %
JavaPurse	13410 bytes	791 bytes	5.90 %
Wallet	3560 bytes	140bytes	3.93 %
Connetction Manager	3440bytes	59bytes	1.71 %
Total	27591bytes	1200bytes	4.35%

표 3. 검증기들의 성능 비교

Verifier	Size of Code	Automatic Proofs	Overhead
Our Verifier	5 KB	50 ~ 60 %	4 ~ 5 %
TL Verifier	10 ~ 15 KB	70 % ~	0 ~ 2 %
Formal Verifier	40 ~ 100 KB	70 ~ 80 %	0 ~ 5 %
General Verifier	300 ~ 500 KB	80 ~ 100 %	0 %

5. 결론

본 논문에서는 정적인 체크 방식을 기반으로 한 효율적인 알고리즘으로 구성되어진 검증기에 관해서 제안하였다. 검증기는 USIM칩과 같은 자원 제약적인 디바이스에 알맞도록 설계되어졌으며 Proof Componet의 사이즈 축소와 효율적인 방법으로 많은 바이트 코드들을 검증할수 있도록 연구 되어졌다. 기존에는 스마트카드 상에서의 검증기 탑재가 불가능 하다고 생각했지만, 본 논문에서 연구한 소형 검증기는 적은 크기를 위해 효율적인 알고리즘을 사용하여 충분히 소형기기에 탑재될 만큼 크기를 줄인 것이기 때문에 탑재가 가능하며 검증기를 통해서 기기의 보안성, 효율성, 안정성을 더욱 증대 시킬 것이다.

참고문헌

[1] Z. Chen, Java Card Technology for Smart Cards: Architecture and Programmer's Guide, The Java Series, Addison-Wesley, 2000.

[2] JavaCard Forum. http://www.javacardforum.org

[3] JavaCard Technology. http://java.sun.com/products/javacard

[4] G. Barthe and S. Stratulat. Validation of the JavaCard Platform with Implicit Induction Techniques, volume 2706 of Lecture Notes in Computer Science, pages 337-351, 2003.

[5] G. Barthe, G. Dufay, L. Jakubiec, B. Serpette, S. Melo de Sousa. A Formal Executable Semantics of the JavaCard Platform, volume 2028 of Lecture Notes in Computer Science, pages 302-319, Springer-Verlag, 2001.

[6] Ludovic Casset. Formal Development of an Embedded Verifier for Java Card Byte Code.

[7] J.B. Cho, M.S. Jung, S.I. Jun. An Efficient Small Sized On-Card Verifier for Java Card, volume 2668 of Lecture Notes in Computer Science, pages 552-561, Springer-Verlag, 2003.

[8] P. Cousot, R. Cousot, Abstract Interpretation: a Unified Lattice Model for Static Analysis by Construction or Approximation of Fixpoints, Proceedings of POPL'77, ACM Press, Los Angeles, California, pages 238-252.

[9] G. Mcgraw, E. Felten. Security Java, John Wiley & Sons, 1999.

[10] G. Necula, *Proof-carrying code*, Proceedings of POPL'97, pages 106-119, ACM Press, 1997.